SigC667x / SigC641x User Guide

Includes Installation and Test for

- **SigC6xxx Hardware [1]**
- **DirectCore® Software**
- **CIM® Software [2]**

Copyright © 2012-2015 Signalogic, Inc.
Revision C1a
Nov 2015

Signalogic, Inc.
9617 Wendell @ Skillman
Dallas, TX 75243
Tel: 214-349-5551
Fax: 214-343-0163

http://www.signalogic.com
tech_support@signalogic.com

[1] Includes SigC667x 32-core and 64-core PCIe cards

[2] Includes Texas Instruments code gen tools installation

TABLE OF CONTENTS

1 SigC667x Hardware Installation 5

1.1 SigC667x Card Overview 5

1.2 Installing SigC667x Cards 6

1.3 SigC667x Card and Firmware Revisions 10

2 SigC641x Hardware Installation 13

2.1 SigC641x Module Overview 13

2.2 Installing SigC641x Modules 14

2.3 SigC641x Loopback Connections 16

2.4 SigC641x Module and Logic Revisions 17

2.5 SigC641x Processor and SDRAM Clock Rates 18

3 Server Installations 19

3.1 HP DL380p Proliant Series 19

3.2 Dell R720 21

3.3 Supermicro 1U 23

3.4 Small Form-Factor Servers 25

3.5 Desktop and Other PC Enclosures 25

3.6 External Power Cables 25

4 Software Installation 29

4.1 Installation Steps 29

4.1.1 Automated Installation Process Notes 30

4.1.2 Manual Install 31

4.2 Building the Loadable Kernel Module (LKM) Driver 35

4.2.1 Old Style Driver Notes 37

4.2.2 Linux Kernel-Devel Build Requirement 38

4.3 Building DirectCore Libraries 41

4.4 Test Programs 44

4.4.1 memTest 45

4.4.2 boardTest 52

4.4.3 fftTest 56

4.4.4 streamTest 59

4.4.5 appTest 64

4.5 Texas Instruments Tools Installation 97

4.5.1 TI Code Generation Tools Download 97

4.5.2 TI RTOS-MCSDK Download 100
4.5.3 TI SYSBIOS, IPC, and XDCtools Download 102
4.6 RTAF Software Usage 103
4.6.1 RTAF Configuration Options 103
4.6.2 RTAF Peripheral Support 104
4.6.3 RTAF Development and Host Platform Notes 104
4.7 CIM® Hyperpiler™ Software 108
4.7.1 Running User Programs with CIM 108
4.7.2 vid_streaming demo 109
4.7.3 Convolution demo 113

5 pn4Test Host Program 117

6 pn4Test Theory of Operation 124

6.1 MCBSP Test Mode 127
6.2 UART Test Mode 127
6.3 GPIO Test Mode 128
6.4 Host Operation 129

Revision History

Created	Rev A	Jul 2012	
Modified	Rev B	Nov 2012 – Jan 2013	Merged SigC677x documentation
Modified	Rev B2	Feb 2013	Updated with software changes
Modified	Rev B3	Feb 2013	Added server installations and initial CIM software sections updated with hardware revision and software changes
Modified	Rev B4	Mar 2013	Added video encoding test sections
Modified	Rev B5	Jun 2013	Added RTAF software usage section and updated memTest
Modified	Rev B6	Aug 2013	Added apptest and call test procedures
Modified	Rev B7	Nov 2014	Added fftTest and videoTest DirectCore demo program sections, CIM software install and build details, and CIM® Hyperpiler™ demo section
Modified	Rev B8	Jan 2015	Added 64-core card install instructions, target software install and build instructions, additional driver build notes, and additional library build notes
Modified	Rev B9	Feb 2015	Added additional 64-core card documentation and updated streamTest software demo information
Modified	Rev B9a	Apr 2015	Additional card revision information
Modified	Rev C1	Nov 2015	Added automated install information
			Removed videoTest reference and replaced with streamTest
			Added CIM® Hyperpiler™ demo information for vid_streaming and convolution demos
Modified	Rev C1a	Dec 2015	Added more information about Hyperpiler demos and how to implement image analytics in the streamTest demo

1 SigC667x Hardware Installation

The sections below cover installation of specific SigC66x and SigC64x card types. Note that section 3, Server Installations, covers a few different types of popular servers and may have some additional hardware installation information that could prove useful, depending on your type of server.

1.1 SigC667x Card Overview

The SigC6678-32 and SigC6678-64 are 32-core and 64-core Gen2 and Gen3 PCIe cards, respectively. These cards combine high performance CPU cores with high bandwidth PCIe interface and large amounts of memory per core, both shared between cores and dedicated external core memory. From the top of the card, the CPU devices can be seen as shown in Figures 1-1a and 1-1b below.

***Note:* Production cards have a heat sink and embedded fan that cover most of the top surface of the card.**

Figure 1-1a, SigC6678-32 top view, without heat sink

Figure 1-1b, SigC6678-64 top view, without heat sink

The SigC6678-32 card has one fan and the SigC6678-64 card has two fans. Including the heat sink and fan(s), the overall card thickness remains within the single-slot height limit given in the PCIe specification.

1.2 Installing SigC667x Cards

To install the SigC6678-32 and SigC6678-64 cards, they should be inserted in a x8 or larger Gen2 or Gen3 PCIe slot. An example of a SigC6678-32 card installed vertically in a x16 PCIe slot is shown in Figure 1-2 below.

Figure 1-2, SigC667x installed in x8 or larger PCIe slot

Cards may also be installed horizontally, depending on the riser configuration with the host server. Additional airflow from the server is typically not required due to the self-contained heat sink and fan. Maximum power consumption is about 54W for the 32-core card, and about 110W to 120W for the 64-core card.

> ***CAUTION! Power should be fully removed from the motherboard prior to card installation. Be sure that the card is fully seated and aligned (upon close inspection, no edge connector fingers on the card should be visible above the edges of the PCIe slot), the backplate on the card is not obstructed, the card is fully seated and flat relative to the PCIe connector, and that any part of the card that overhangs the PCIe connector is not touching components on the motherboard, for instance heat sinks or tall capacitors.***

The SigC667x cards support two boot modes: emulation boot mode and I2C boot mode. Boot mode is selected by Switch 1 (labeled "SW1" on the card), as shown in Figure 1-3 below. I2C boot mode is the default setting and should be used for normal SigC667x card operation. Emulation boot mode is reserved for low-level card testing and debug, and is sometimes known as "JTAG debug".

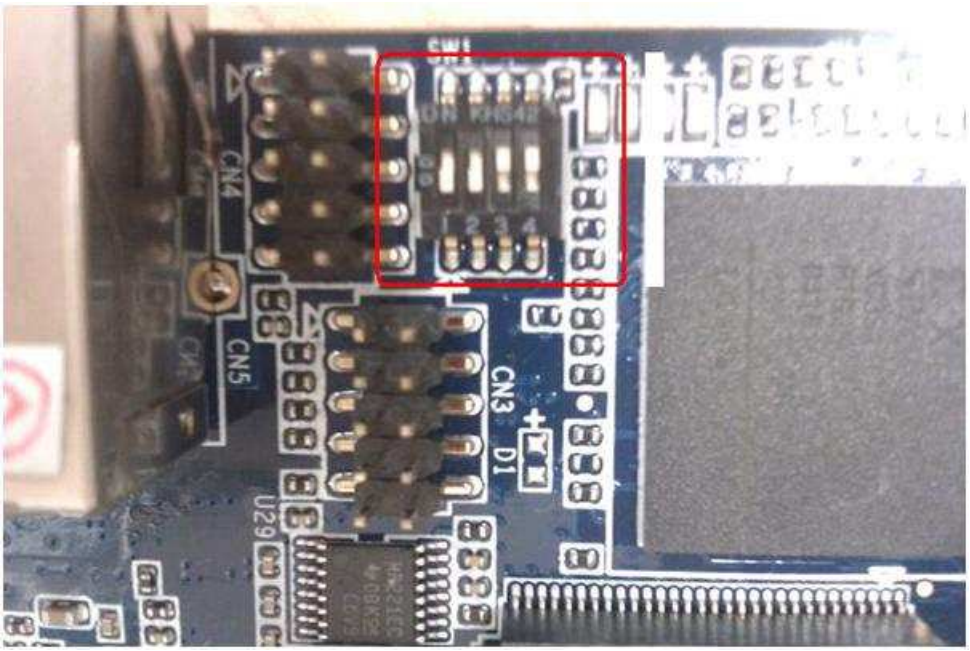
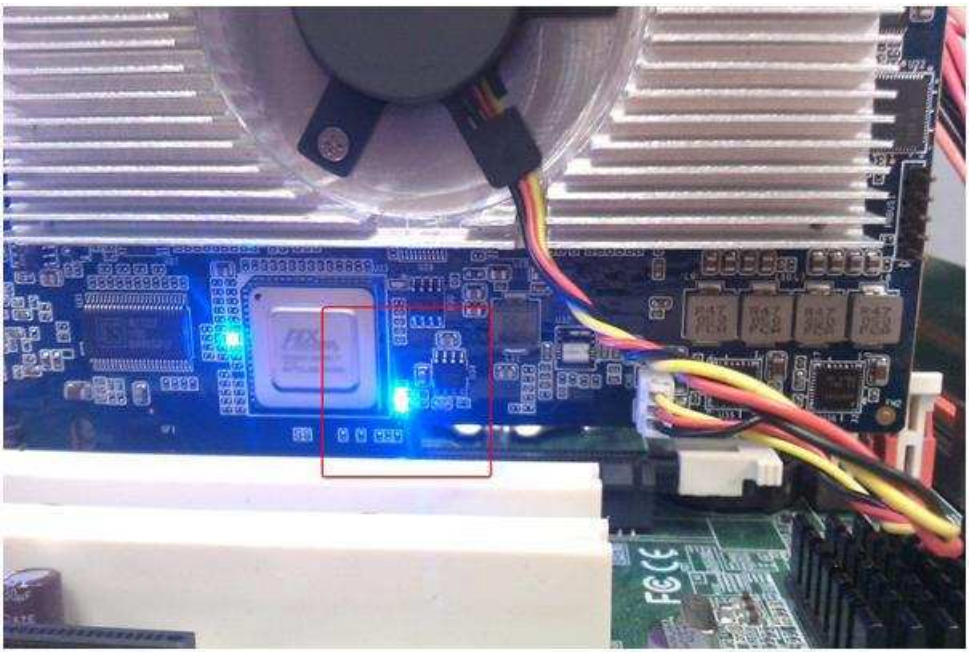


Figure 1-3, SigC6678-32 card SW1 (boot mode)

CAUTION! ***It is a known issue that in some servers SigC667x cards with A101 revision firmware may not complete their on-card I2C boot process before the server completes its internal BIOS scan of the PCIe device tree (which includes PCIe slots). On the SigC6678-32 card there are four (4) "boot LEDs" that will flash to indicate valid PCIe initialization after boot. The placement of these LEDs is shown in Figures 1-4 and 1-5, as shown below.***



 Copyright © Signalogic 2012-2015

Figure 1-4, SigC6678-32 card PCIe boot LEDs, top side

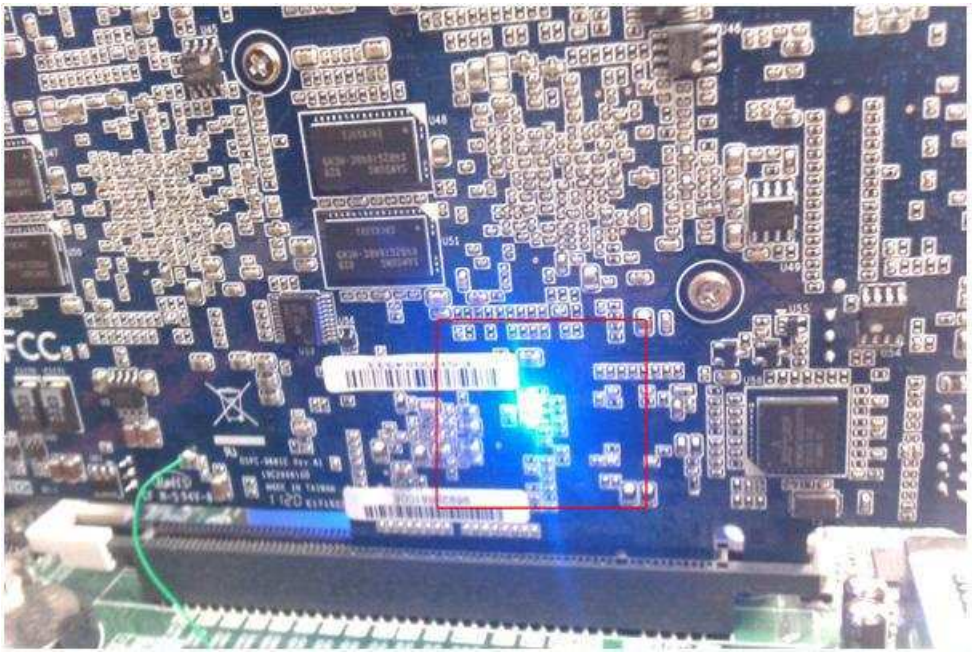


Figure 1-5, SigC6678-32 card PCIe boot LEDs, bottom side

***CAUTION!* If after boot all PCIe LEDs are not flashing, then a system reset, either through software (for example 'shutdown –r now' Linux command) or front-panel switch, can be performed. This is not an issue on cards with A102 revision firmware.**

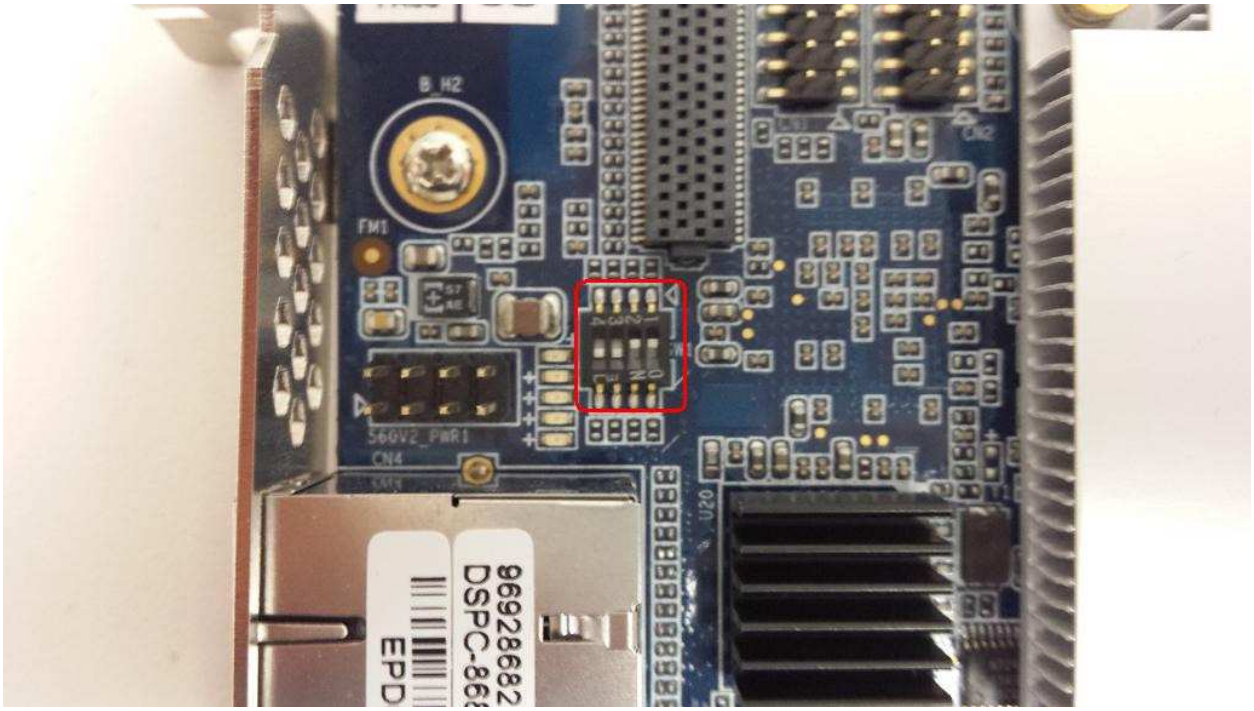


Figure 1-6, SigC6678-64 card DIP switch default settings, top side (highlighted in red)

The SigC6678-64 card DIP switch (labeled "SW1" on the card) should be set as shown in Figure 1-6 above (switch positions 1 and 2 off, and positions 3 and 4 on). If these switch settings are not correct, then the card may fail to boot properly.

Figure 1-7, SigC6678-64 LEDs, top side, after boot (highlighted in red)

After boot, the five (5) LEDs shown in Figure 1-7 above should all be illuminated blue in a "steady on" state, with no blinking (see red highlighted area in the Figure 1-7 image). One LED indicates correct power/voltage conditions on the card, and four (4) LEDs indicate the card's low-level controller FPGA has booted and initialized correctly. As a note, depending on the server form factor, the SigC6678-64, when inserted, may be positioned where the lights and fans are not shown from the top.

CAUTION! If the SigC6678-32 card is inserted horizontally, then make sure the card is flat and is well supported, and does not "sag" or "droop", otherwise the card can develop a permanent warp, which can damage the soldering attachment of chips on the card. For example, if the card is installed on the second slot on a vertical riser, and the PCIe expansion area does not have an internal bracket to support the far end of the card, then it's advisable to insert small pieces of anti-static foam under the card, every 1" to 2", and at the edges of the card, to help support it.

1.3 SigC667x Card and Firmware Revisions

For SigC6678-32, card and firmware revision is printed on the back side of the card, near the inner edge of the PCIe connector, as shown in Figure 1-6 below:

 Copyright © Signalogic 2012-2015

Figure 1-6, SigC6678-32 card and firmware revision information

For SigC6678-64 cards, card and firmware revision information is printed on the back side of the card, as shown in Figure 1-7 below:

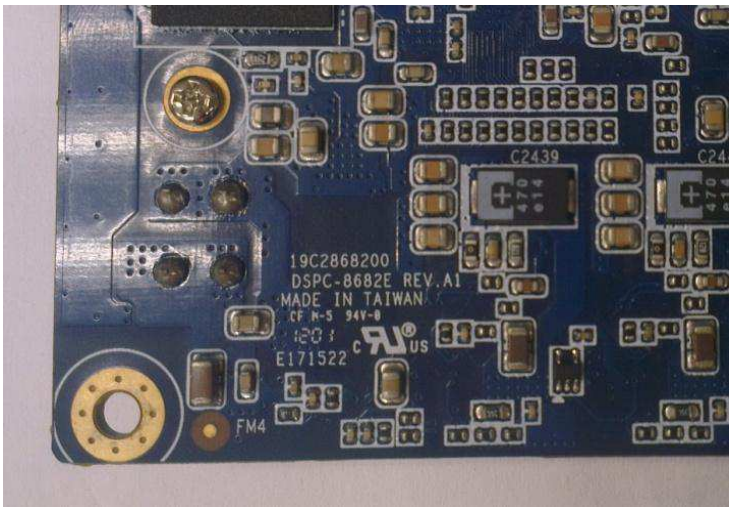


Figure 1-7, SigC6678-64 card and firmware revision information

 Copyright © Signalogic 2012-2015

Card revision information is given in Table 1-1 below:

Hardware Version Information	**Card Type**	
	Revision **CPU Clock Rate, DDR3 Mem Size**	
Silkscreen labeling on back of card	**SigC6678-32**	**SigC6678-64**
9692868100E or 9692868100	A101 1 GHz [1], 1 GB [2]	
9692868101E	A102 1 GHz, 1 GB	
9692868102E	A103 1 GHz, 1 GB	
9692868122E	A103 1.25 GHz, 2 GB	
19C2868200		A101 1 GHz, 1 GB
19C2868201-01		A102 1 GHz, 1 GB
19C2868202-01		A103 1 GHz, 2 GB
19C2868222-01		A103 1.25 GHz, 2 GB
19C2868203-01		A104-1 1.25 GHz, 2 GB

[1] GByte
[2] GHz, or cycles per sec

Table 1-1, SigC667x card and firmware revision information

In the above table, note that silkscreen labeling includes information about firmware and hardware revision, CPU clock rate, and amount of DDR3 memory per CPU.

2 SigC641x Hardware Installation

2.1 SigC641x Module Overview

The SigC6415T PMC module has eight C6415T single-core devices and a PCI interface. From the top of the card, two C6415T devices can be seen as shown in Figure 2-1 below.

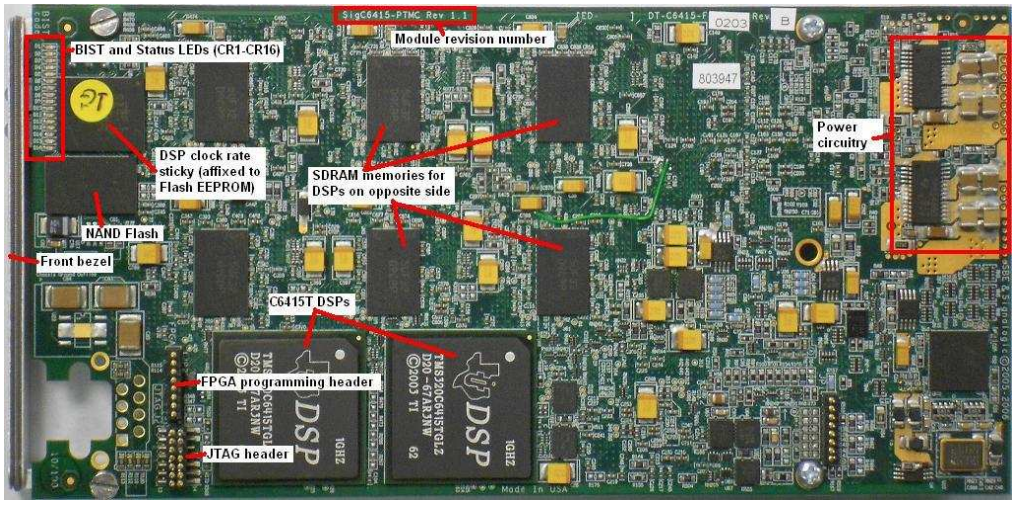


Figure 2-1, Top view of SigC6415T PMC module

From the bottom of the card, the remaining six C6415T devices, PCI interface logic (FPGA), PN4 interface logic (FPGA) and PMC interface connectors can be seen. The PN4 connector is at the upper right as shown in Figure 2-2.

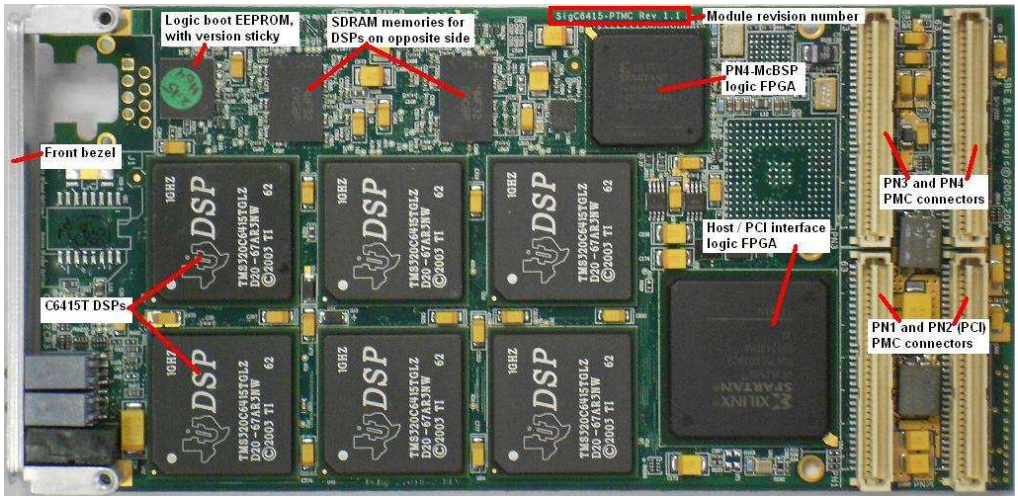

Logic boot EEPROM, with version sticky
SDRAM memories for DSPs on opposite side
Module revision number
PN4-McBSP logic FPGA
Front bezel
PN3 and PN4 PMC connectors
Host / PCI interface logic FPGA
C6415T DSPs
PN1 and PN2 (PCI) PMC connectors

Figure 2-2, Bottom view of SIGC6415T PMC module

 Copyright © Signalogic 2012-2015

2.2 Installing SigC641x Modules

In order to use a SigC641x PMC module, it must first be inserted onto a PMC-to-PCIe adapter card. The module has been tested with a Dynamic Engineering PMC to PCIe adapter (PCIeBPMCX1), shown in Figure 2-3, and an Integrative Innovation PMC to PCIe adapter, shown in Figure 2-4. Ensure that a complete connection is made on the PMC interface when inserting the module: press firmly but gently, and press at the edges of the module; i.e. not directly on components. After insertion, check carefully for any gaps around the edges of all four (4) PMC connectors. Also, make sure the module rests completely flat and in parallel with the adapter card, and the module's front bezel fits correctly into the adapter's end-plate cut-out.

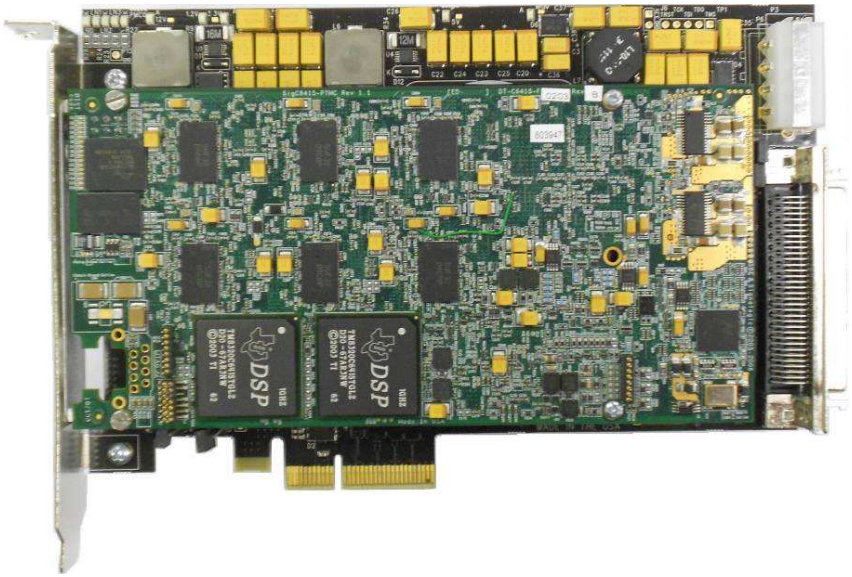


Figure 2-3, SigC6415T module on Dynamic Engineering adapter

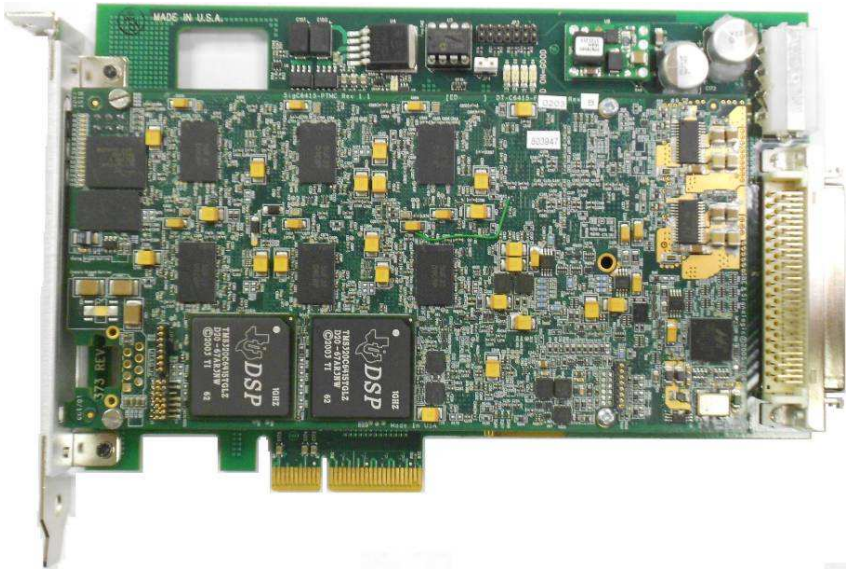


Figure 2-4, SigC6415T module on Integrative Innovation adapter

 Copyright © Signalogic 2012-2015

After inserting the module into an adapter, the adapter card can now be placed in a PCIe slot (or PCI slot if a PMC to PCI adapter is used instead). Ensure that the card is inserted into the PCIe slot with the proper orientation. The bezel plate should be on the edge of the motherboard with the 4-wire internal power connector towards the interior. The card installed into a PCI express slot using a Dynamic Engineering adapter is shown in Figure 2-5. In the case of using a PCI-X adapter in a PCI slot, make sure that the extra PCI-X connector section does not hit anything on the motherboard. Make sure that the 4-wire internal power connector is inserted before turning on the system.

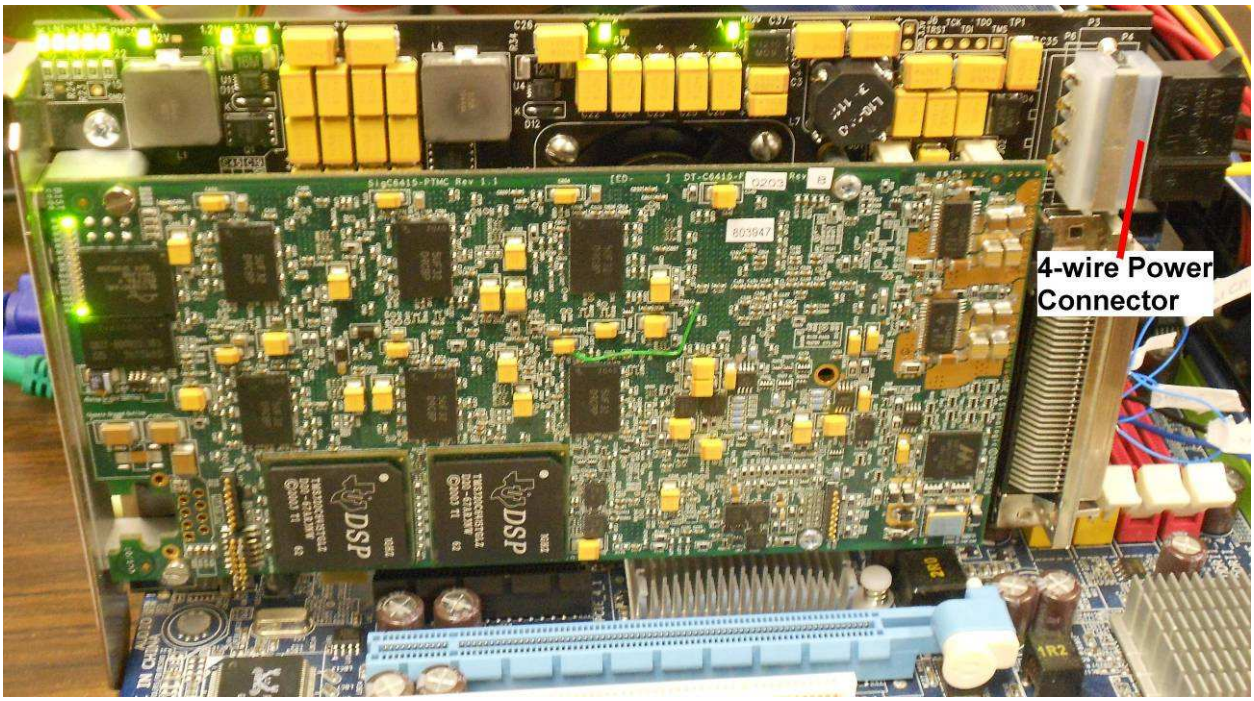


Figure 2-5, SigC6415T module on adapter board, inserted into PCIe slot

 Copyright © Signalogic 2012-2015

2.3 SigC641x Loopback Connections

In order for test software to operate correctly, loopback connections on the correct PN4 and backplate connector pins should be established (see Figures 6-1 and 6-2 for PN4-to-backplate connector mapping diagrams). In the case of the Dynamic Engineering adapter, wires can be inserted into the connector holes to connect the corresponding pins of the SCSI connector, shown in Figure 2-6. For the Integrative Innovation adapter, a connector with these loopback connections should be made to be attached to the alternate SCSI connector on this adapter, shown in Figure 2-7.

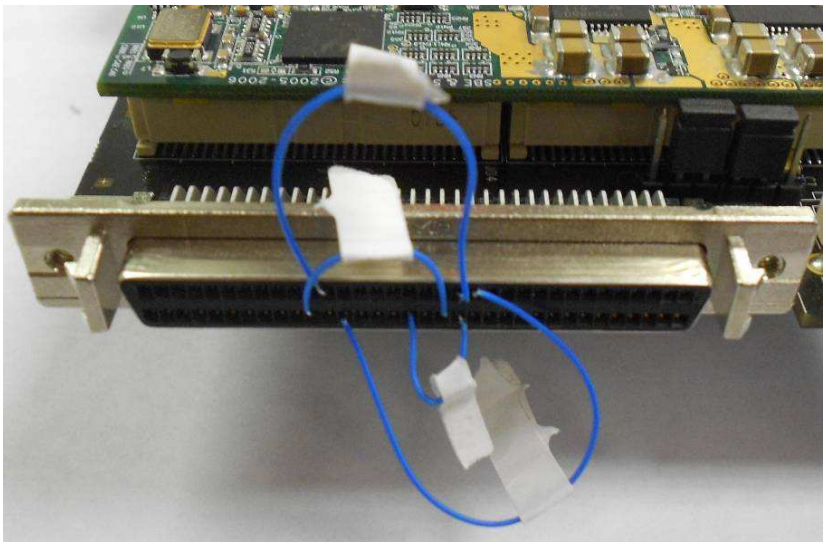


Figure 2-6, Dynamic Engineering adapter loopback wires, inserted into P2 external I/O connector

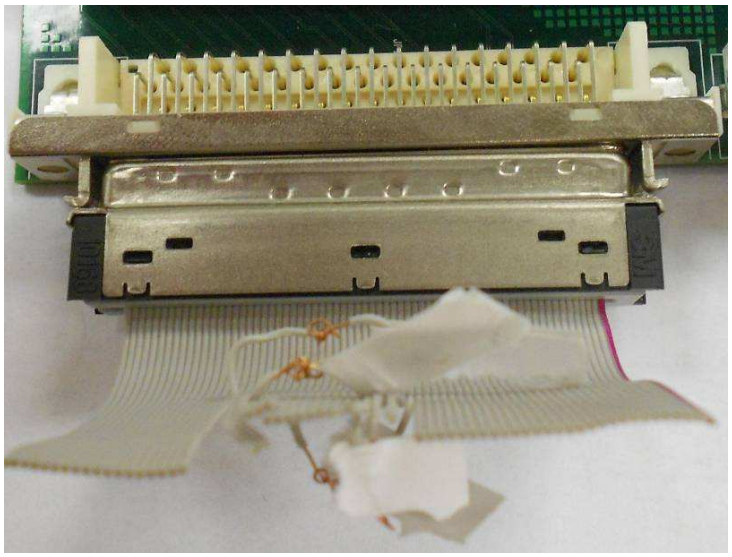


Figure 2-7, Integrative Innovation adapter loopback connector, mated with JP2 external I/O connector

 Copyright © Signalogic 2012-2015

2.4 SigC641x Module and Logic Revisions

The module revision number is printed on the silkscreen along the left edge (with the bezel plate towards you) of both sides of the module as shown in Figure 2-8. The logic revision number is affixed to the Xilinx boot PROM (U39) as shown in Figure 2-9.

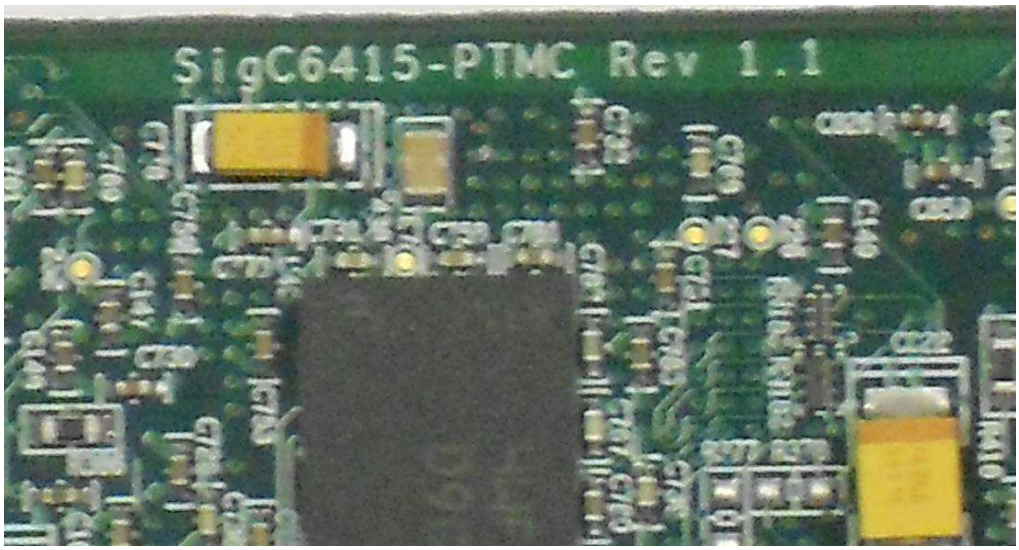


Figure 2-8, SigC641x module revision number

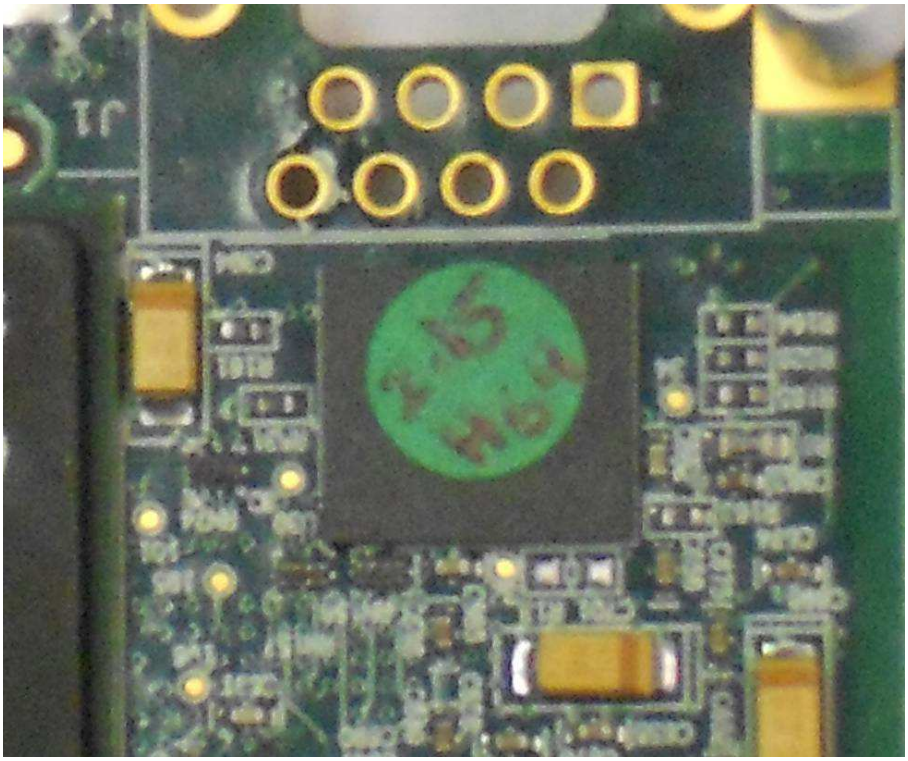


Figure 2-9, SigC641x logic revision number

 Copyright © Signalogic 2012-2015

2.5 SigC641x Processor and SDRAM Clock Rates

The processor clock rate for the particular module used is affixed to the flash memory chip (marked U8 on the module silkscreen) which is located next to the module status LEDs (marked D1-D16 on the module silkscreen) and is shown in Figure 2-10 below. The clock rate can also be determined from the part markings on the C6415T processors (e.g. 7E3 for 720 MHz and 1GHZ for 1 GHz). The SDRAM clock rate is the processor clock rate divided by 6 which would be 120 MHz for 720 MHz processors and 166.67 MHz for 1 GHz processors.

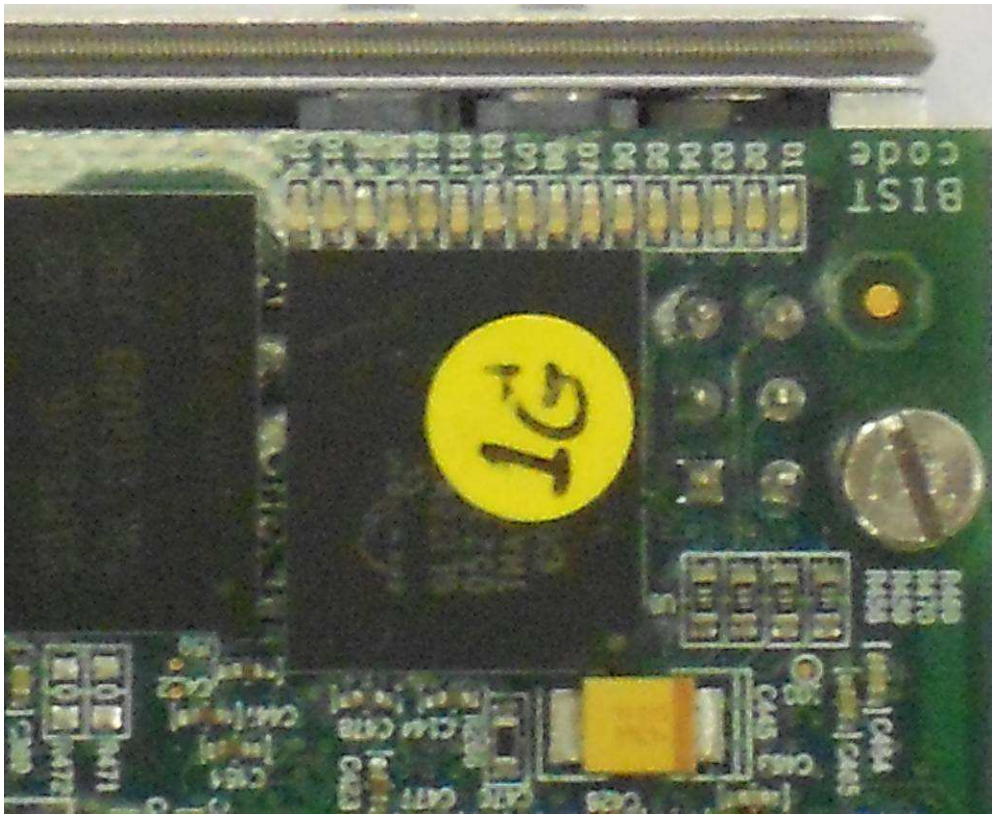


Figure 2-10, SigC641x processor clock rate module labeling

 Copyright © Signalogic 2012-2015

3 Server Installations

The Server Installations section contains information on specific types of servers. This information does not cover all servers, but some of the physical and configuration procedures and requirements documented here may also apply to other server enclosures.

3.1 HP DL380p Proliant Series

Figures 3-1a and 3-1b below show a SigC6678-64 card installed in an HP DL380p Gen8 server. Note the power cable required to connect the card to the 12V connector on the riser.

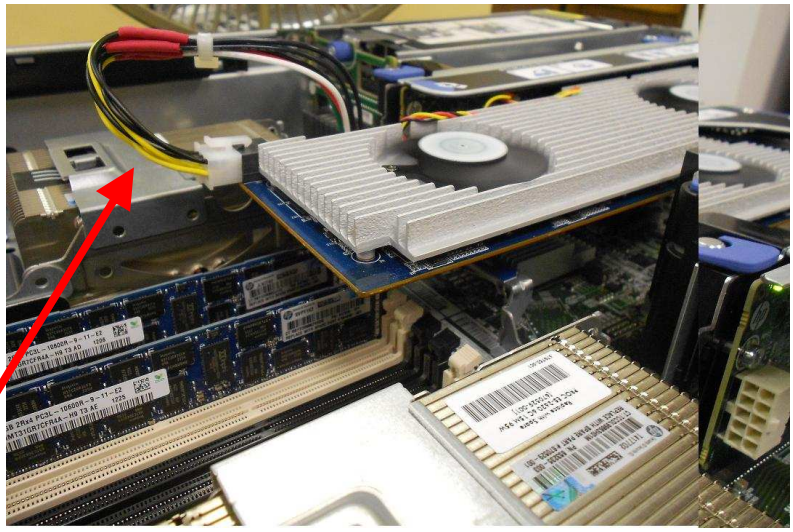


Figure 3-1a, SigC6678-64 card installed in DL380p server, side angle view

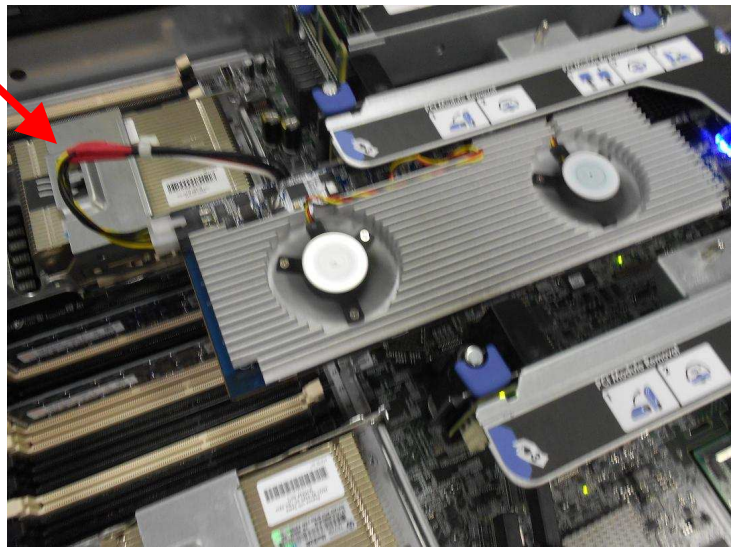


Figure 3-1b, SigC6678-64 card installed in DL380p server, top view

 Copyright © Signalogic 2012-2015

An additional power cable is required for cards with a PEG (PCI Express Graphics) type of connector, which has become a de facto industry standard due to the proliferation of PCIe GPU boards. SigC6678-32 cards do not have a PEG connector, as their power consumption is limited to about 50W. The HP part number of the power cable shown in Figures 3-1a and 3-1b above is:

```
P/N 504660-003
```

This HP cable is also shown in Figure 3-7 below (in the picture, the HP cable is on the left, with a bar code tag).

 Copyright © Signalogic 2012-2015

3.2 Dell R720

Figure 3-2a below shows two (2) SigC6678-64 cards installed in a Dell R720 2U server. Note the 12V power cables connecting cards and risers (Dell P/N 490-13616, also may be available through online distribution using P/N 9H6FV).

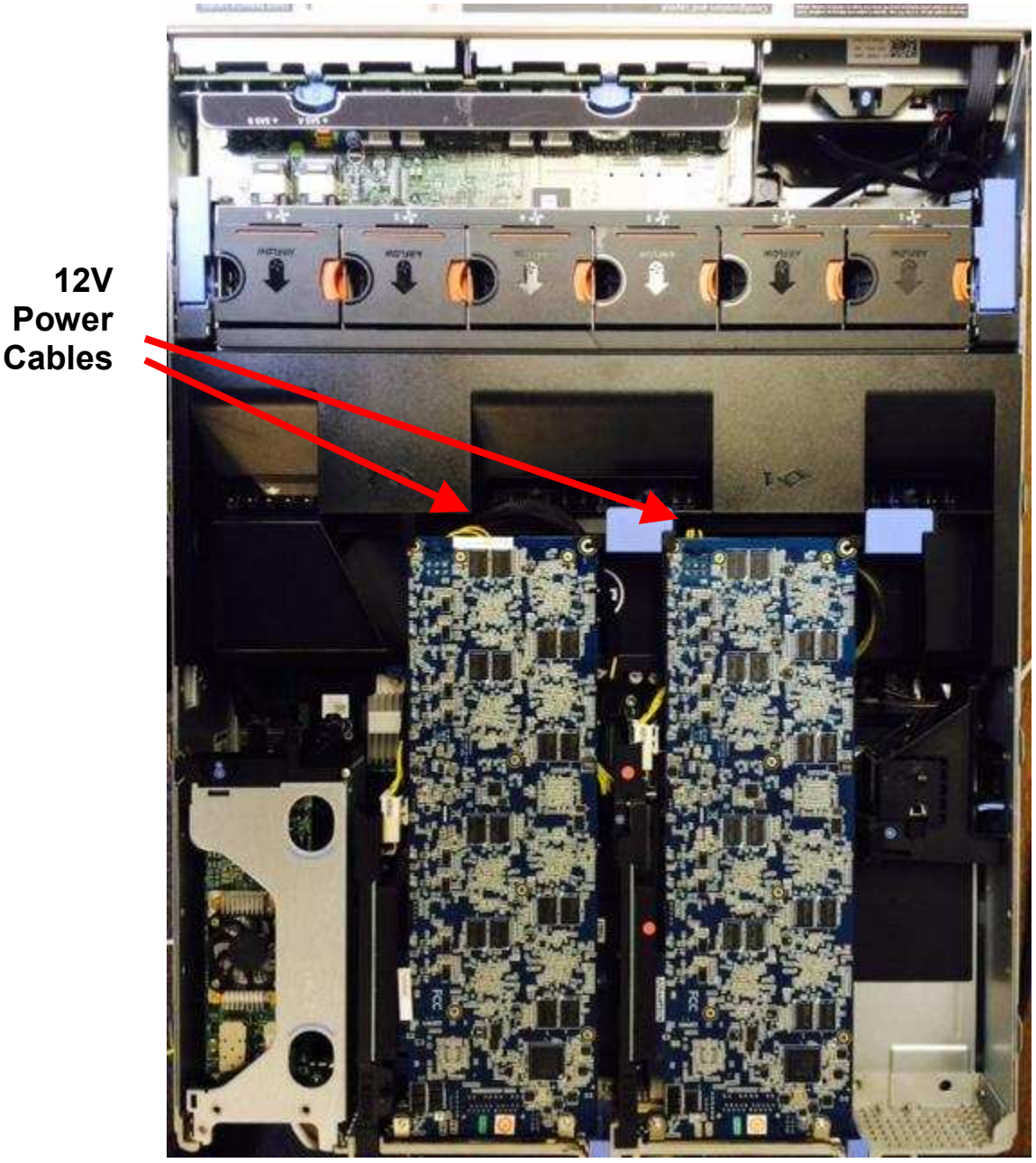


Figure 3-2a, Two (2) SigC6678-64 cards installed in a Dell R720 1U server (top view)

An additional power cable is required for cards with a PEG (PCI Express Graphics) connector, which has become a de facto industry standard due to the proliferation of PCIe GPU boards.

 Copyright © Signalogic 2012-2015

SigC6678-32 cards do not have a PEG connector, as their power consumption is limited to around 50W. The Dell part number of the power cable shown in Figure 3-1a above is:

P/N 504660-003

Figure 3-2b below shows the SigC667x-64 card power cable adapter used in Dell R720 and R730 servers:

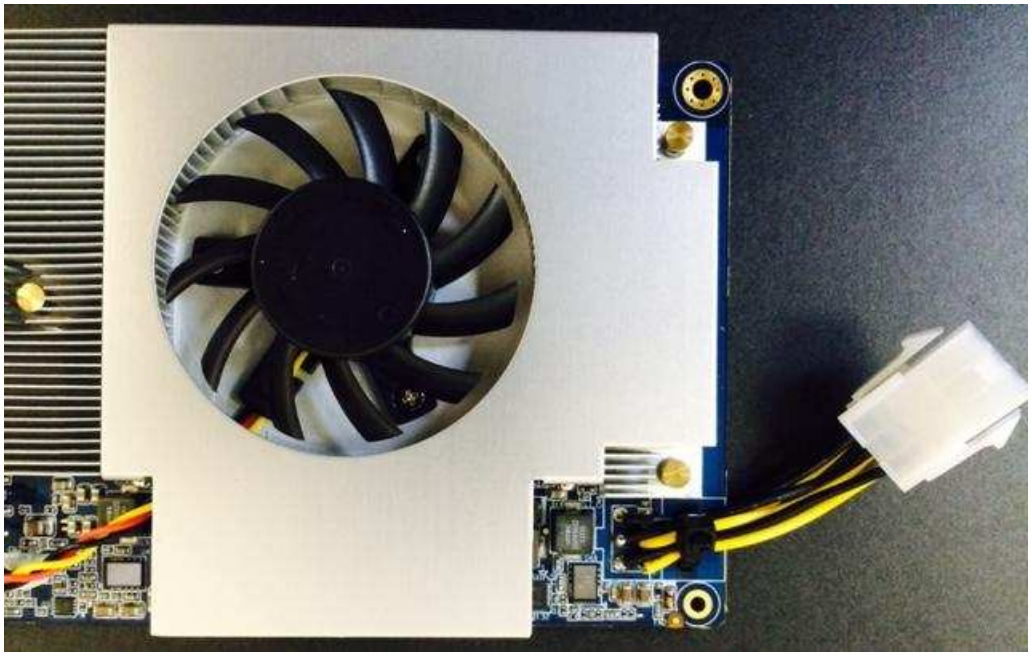


Figure 3-2b, SigC6678-64 card power cable adapter

The power cable adapter mates to the Dell 12V power cable (P/N listed above). Note that in Figure 3-2a above, the power cable adapter is not visible as the cards are installed with component side down.

 Copyright © Signalogic 2012-2015

3.3 Supermicro 1U

Figure 3-3 below shows four (4) SigC6678-32 cards installed in a Supermicro 1U server. In this case, an unusual narrow motherboard and "dual slot riser" configuration makes it possible to insert PCIe cards on both sides of the server. Note that for SigC6678-32 cards, no additional power cables are used.

Figure 3-3, Four (4) SigC6678-32 cards installed in a Supermicro 1U server, top angle view

Figure 3-4 below shows a SigC6678-64 card installed in a 1U server. The card length is about 12.2", which requires a full-length PCIe slot. In this example, the riser card (see annotation in the picture) has two (2) PCIe slots, which allows two (2) SigC6678-64 cards to be installed.

 Copyright © Signalogic 2012-2015

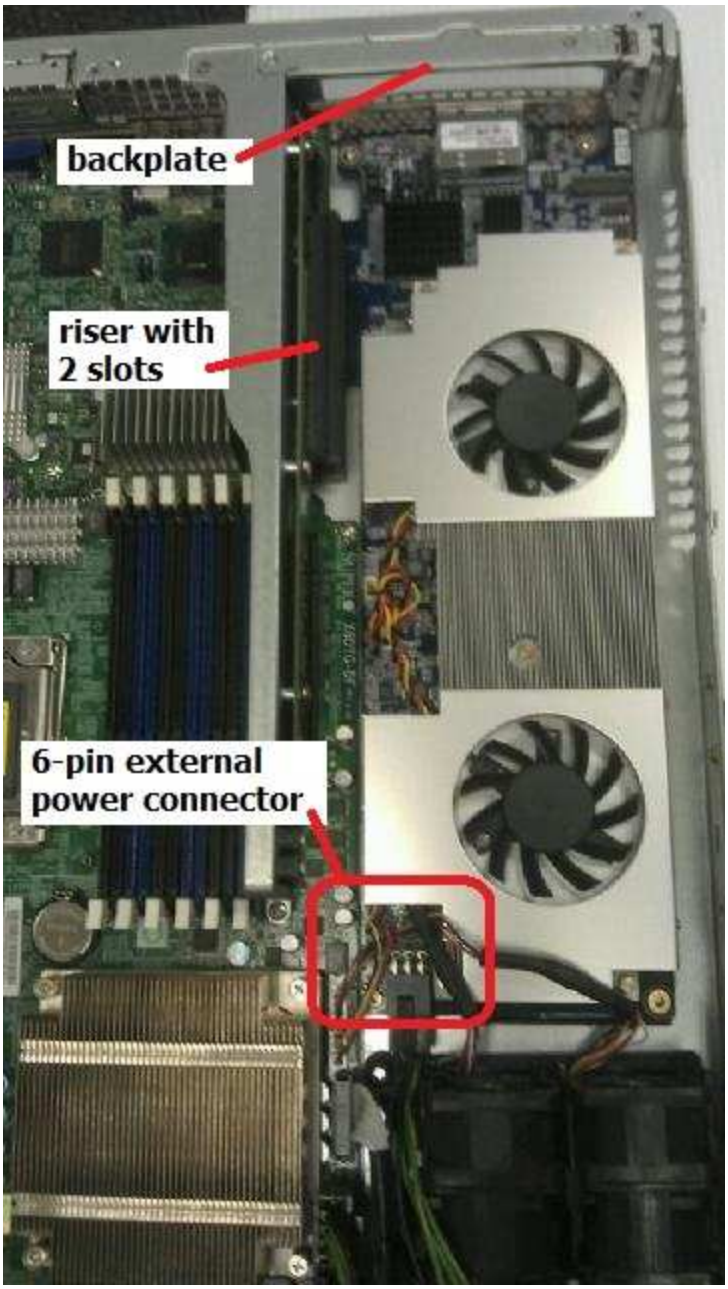


Figure 3-4, SigC6678-64 card installed in a 1U server, top view, with 6-pin external power connection highlighted (red square)

Note that it is also possible to "mix and match" cards, for example, one SigC6678-32 card and one SigC6678-64 card installed on the same riser, or otherwise installed in different PCIe slots in the same server.

 Copyright © Signalogic 2012-2015

3.4 Small Form-Factor Servers

Installation for small form-factor servers, such as mini-ATX, is similar to server installation. Figure 3-5 below shows a dual-core x86 Atom CPU motherboard and a SigC6678-32 card, resulting in an enclosure size of about 8” x 10” x 4” and total power consumption of around 80 W.

Figure 3-5, SigC6678-32 card installed in a mini-ATX server

3.5 Desktop and Other PC Enclosures

Installation for non-server PC enclosures, for example desktop, tower and shuttle cases, is very similar to server installation, with the exception of external power cables for SigC6678-64 cards. Note that for SigC6678-32 cards, no additional power cables are used. See the following sections for more information on external power connection.

3.6 External Power Cables

Figures 3-6 and 3-7 below show different external power cables that can be used with SigC6678-64 cards, for example in "tower" or other traditional PC box enclosures. Note that for SigC6678-32 cards, no additional power cables are used.

 Copyright © Signalogic 2012-2015

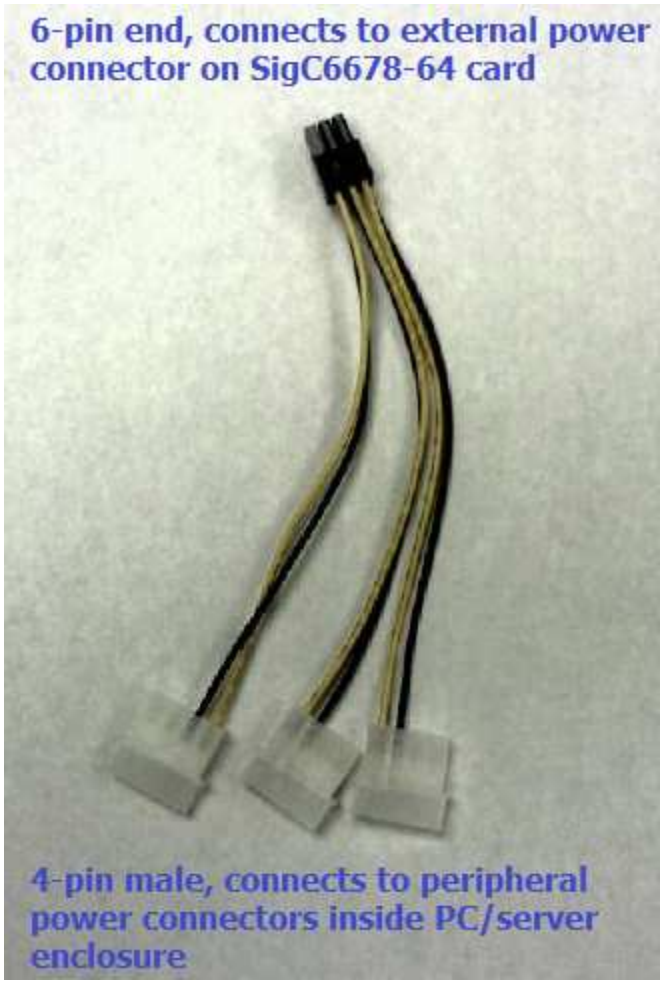

6-pin end, connects to external power connector on SigC6678-64 card
4-pin male, connects to peripheral power connectors inside PC/server enclosure

Figure 3-6, External power cable for SigC6678-64 cards that connects to standard 4-pin peripheral power cables inside a desktop or other non-server form-factor PC enclosure

Figure 3-6 above shows an external power cable that mates with standard 4-pin power cables found inside a desktop or other PC enclosure that provide power to peripherals, such as hard disk drives, DVD players, etc. For the SigC6678-64 card, a minimum of two connectors should be mated, but it is preferable if all three connectors are mated (see notes below).

Figure 3-7 below shows external power cables that mate with a server internal riser card.

Note that typically, yellow wires are +12V and black wires are Gnd. Additionally, some server vendors use proprietary cables connecting to their risers. Use caution when sourcing cables, as a generic cable may cause damage to a system.

 Copyright © Signalogic 2012-2015

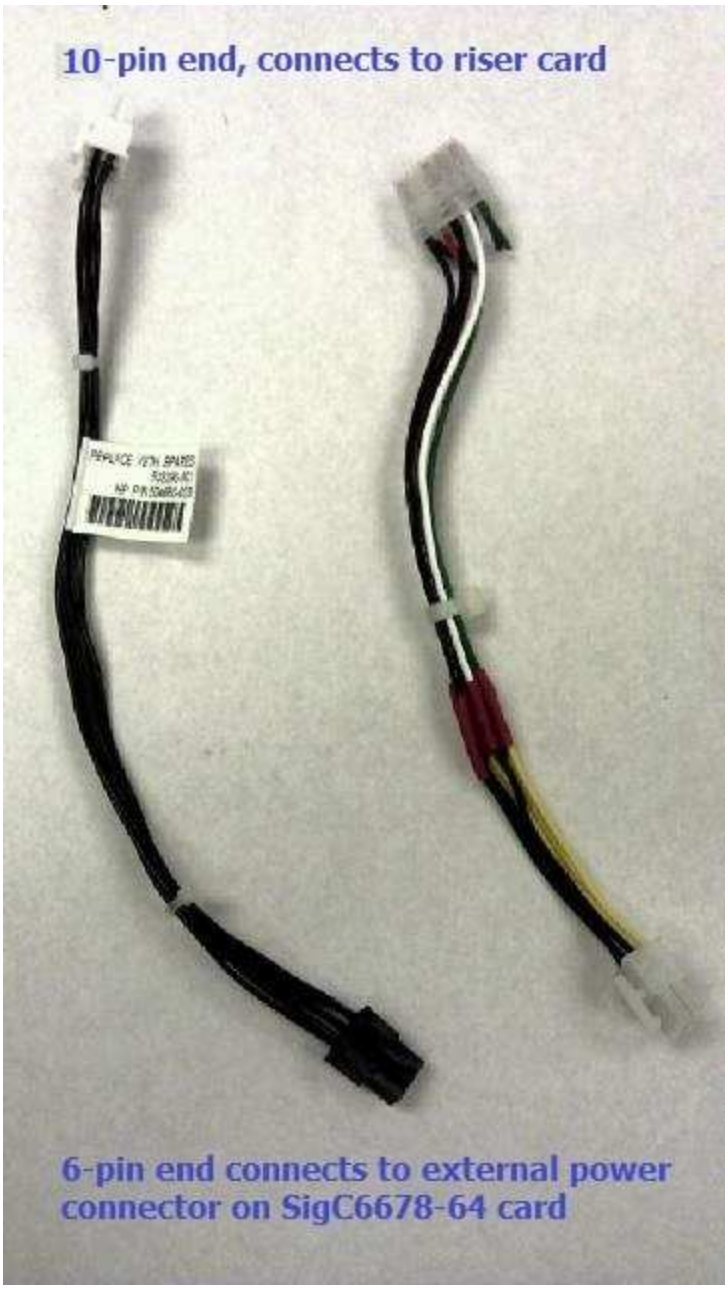


Figure 3-7, External power cable for SigC6678-64 cards that connects to an internal riser card

If for any reason you are unsure about which connector to use, and you want to absolutely verify voltages prior to card installation, a simple test step is to attach the power cable to the server or (or other PC enclosure) internal connector, without the SigC6678-64 card installed and with the top panel or lid open on the enclosure, boot the machine, and use a voltmeter (set to DC volts) to measure the pins. Measured voltages for the external power cable should match up to the pin assignment shown in Figure 3-8 below:

 Copyright © Signalogic 2012-2015

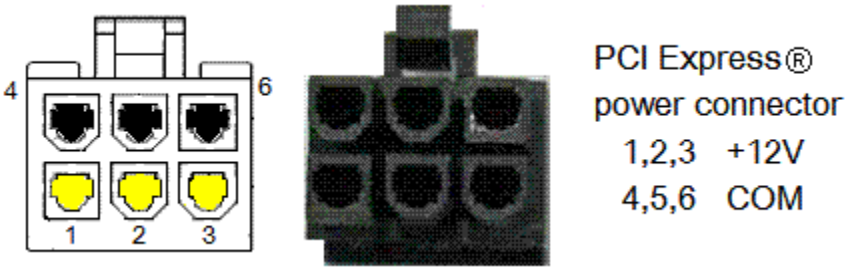


Figure 3-8, 6-pin external power cable pin assignment

Figure 3-8 above shows voltage pin assignment for the 6-pin end of cables shown in above figures and pictures. Notes about Figure 3-8:

- View is facing the connector, i.e. as if you're "standing" on the card, looking at the end of the cable

- COM means Common, or Gnd

Some cables may "break out" the 6 wires, for example Figure 3-6 above shows the 6 wires distributed among 3 peripheral power connectors, in 2-wire pairs (one +12V wire and one Gnd wire for each pair). For the SigC6678-64 card, at least 4 wires (2 yellow, 2 black) should make electrical connection

These connectors are sometimes called PEG connectors (PEG = PCI Express Graphics). Using this terminology may help when ordering cables and/or connectors

4 Software Installation

Signalogic software consists of (i) Linux host software and (ii) target software and build tools. "Host" in this case means the server, ATCA, embedded system, or other platform / enclosure that hosts the "target". Host and target can be physically interfaced in multiple ways:

- Host server shared memory via PCIe bus, for example PCIe cards containing one or more multicore CPUs or other compute devices, such as neural net chips. The SigC6678-32 and SigC6678-64 are accelerator cards containing:
- Network connection
- Other methods, such as SRIO

Note the definition of host and target is flexible as there is not always a clear physical boundary. For instance an SoC device might contain one or more ARM cores and one more compute intensive cores; in this example the ARM cores would be the host and the compute cores the target. As another example, inside an x86 server, some x86 cores might be "control plane" (host running Linux) and others "data plane" (target).

4.1 Installation Steps

To install Signalogic software, perform the following steps:

1. First, download distribution files from secure page, USB key, or other source as provided. Distribution files include several .rar files and an install script. The table below shows .rar file types. Note that depending on which software you have licensed, have on trial, or otherwise have obtained, you may not have all .rar files.

.rar Files	Type	Description / Comments
DirectCore	Host	Includes drivers (both physical and virtIO), shared libraries, and example application test and demo programs
CIM	Host	Includes CIM® Hyperpiler™ pre-processing tools, utilities and scripts, and demo programs
mCPU_target	Target	Includes target (multicore CPU, DSP, neural net, etc) run-time test and demo programs and source code, RTAF components, and CIM components
video_files	Data	Includes .yuv format video files for video compression, streaming, and image analytics demo and test purposes

2. Run the automated install script "autoInstall_Sig_SW.sh", which will prompt following options:

   - Full software installation. On choosing this option, the install script will ask for software installation path and then proceed with installation

- Uninstall. This option can be used for a full software uninstall, and is also useful in case of errors or a need to repeat the installation process.

- Exit

3. The install script will ask for an install path. The default is "/". Other tested paths include /home, /opt/, root.

During the install process you may see some prompts, but you should not get error messages. Upon completion, a Signalogic software folder should be created, with subfolder hierarchy as shown in Figure 4-1 below.

4.1.1 Automated Installation Process Notes

Following are some important notes about the automated installation process.

4.1.1.1 Sudo Root Privilege

At some point the install script will need to temporarily gain sudo root privilege, for example to perform driver configuration, modify config files, etc. When the install script reaches that point it will prompt to perform or skip these steps; if you choose to perform the steps then the script will prompt for a sudo root password.

Note that in Ubuntu, giving a user sudo root privilege can be done by adding the user to the "administrators" group (as shown here http://askubuntu.com/questions/168280/how-do-i-grant-sudo-privileges-to-an-existing-user). In CentOS a user can be added to the "/etc/sudoers" file (as shown here https://wiki.centos.org/TipsAndTricks/BecomingRoot).

4.1.1.2 Dependency Packages

The install script will check for system dependency packages required to complete the installation. If not installed it will install, otherwise it will prompt as follows:

Y (Yes)

N (No)

A (yes to All, no further prompts)

It should be the case that all required dependency packages will be installed, but it is possible that some may not, depending on your system.

4.1.1.3 Automated Build

Depending on the software license and/or delivery method, the install script will either (i) load pre-built binary files or (ii) build from source code DirectCore libraries, drivers, and example / demo applications. Binary files include .so files for libraries, .ko files for drivers, .out files for multicore CPU targets, and executable files (no extension).

4.1.1.4 Driver Load

The install script will load DirectCore drivers and configure the system so that drivers are loaded automatically after a reboot. After successful software installation, you can verify the main DirectCore driver is loaded using the following command:

```
"lsmod | grep sig_mc_hw"
```

which should show output similar to this:

```
sig_mc_hw       pid          0
```

where "pid" is a Process Id number (for example 92357) and the right hand number is the "use count" for the driver (which should be zero just after installation).

4.1.2 Manual Install

If for some reason, the distribution .rar files must be installed manually, then they can be unzipped under an arbitrary path using the following commands, creating a Signalogic software folder, with subfolder hierarchy as shown in Figure 4-1 below. Most files will be installed under the given arbitrary path. Also a "signalogic" configuration file will be installed under:

```
/etc/signalogic
```

Here is a command line example of manual install using the .rar files (actual files dates such as 8Jul15 should appear in the filename instead of DDMMYY):

```
unrar x Signalogic_sw_host_DirectCore_DDMMYY.rar /home
unrar x Signalogic_sw_host_CIM_DDMMYY.rar /home
unrar x Signalogic_sw_target_mCPU_target_DDMMYY.rar /home
```

The above commands will unrar files to the "/home" folder. Note the "x" flag is required to preserve .rar file folder structure and extract using full paths.

Note: The folder structure contained in the .rar files must be preserved. After installation, the resulting folder structure should be as shown in figures 4-1 through 4-3 below.

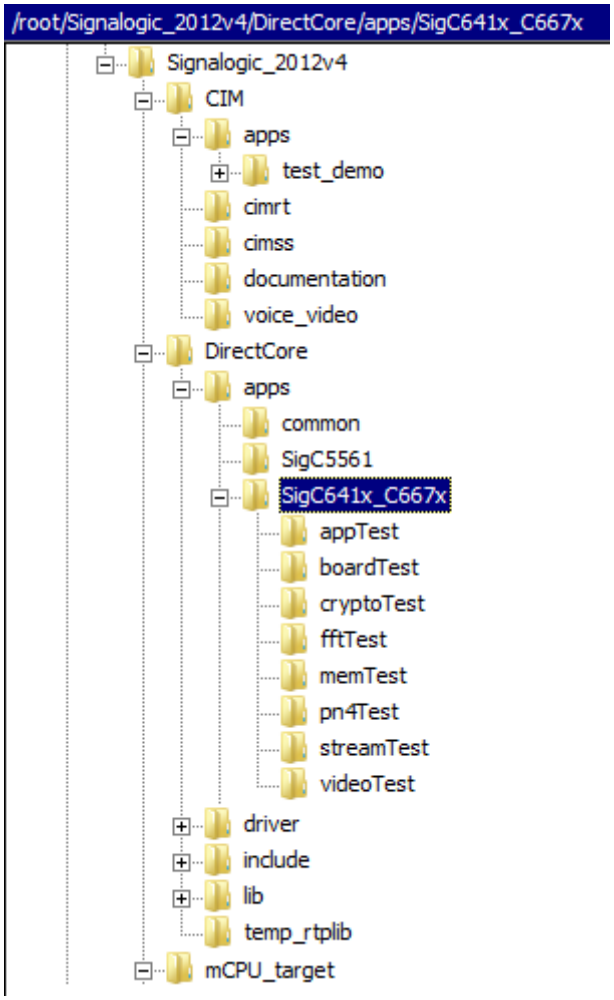


Figure 4-1, Signalogic host and target software folder hierarchies, as they should appear after software installation

Note! ***The physical Signalogic software folder may be suffixed with "_2012v4", "_2014v5", etc, but in all cases a symlink will be created that names the folder as "Signalogic".***

For SigC667x cards and SigC641x cards/modules, a target software .rar file is also provided, which includes software specific to these cards. In general, target software includes run-time test and demo programs (and source code), RTAF components, and CIM components for different types of hardware, such as multicore CPU, DPDK, neural net, DSP, etc. RTAF (Real-Time Algorithm Framework) software provides target software support for host software, including DirectCore software, CIM software, and user-defined projects. The target software .rar file consists of source code and other necessary files to build with Texas Instruments command line code generation (build) tools. For more information about RTAF, see section 4.5 below, RTAF Software Usage. Configuration, project, and other files needed for use with Texas Instruments build tools are also included in the target .rar file.

 Copyright © Signalogic 2012-2015

For SigC667x cards, all TI code generation and configuration is handled either with make files or using the fully automated CIM build process, and intended to be used with TI's command line build tools. The software build process does not interact with TI's Code Composer Studio IDE environment, although that mode is supported if needed due to user preference or application-specific reasons. For target software, the target .rar file installs in the "mCPU_target" subfolder under the main Signalogic folder on a Linux machine, as shown in Figure 4-1 above and Figure 4-2a below.

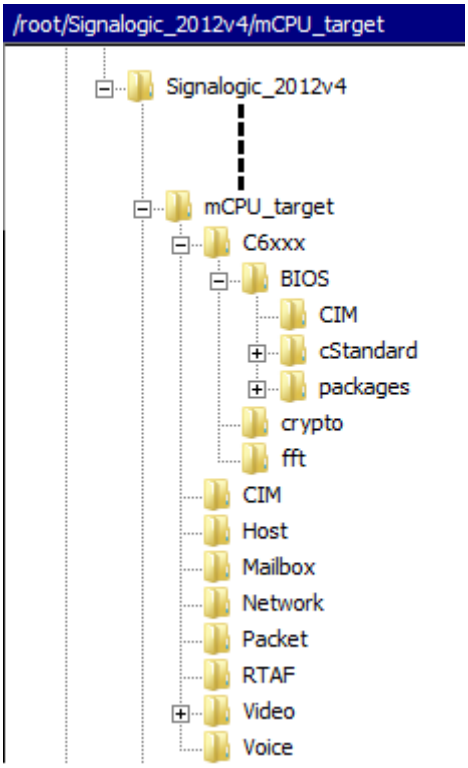


Figure 4-2a, Target software folder hierarchy for SigC667x cards

Note in Figure 4-2a above, RTAF software is stored on the "mCPU_target" subfolder, meaning multicore CPU or other target hardware. This naming convention refers to run-time software for multicore CPU or other target hardware; e.g. CIM accelerator, DSP card, video or telecom card, network processing card, neural net card, etc. Note that various supported multicore CPU and other target hardware types are collectively referred to throughout the User Guide as "targets", and code that runs on the target as executable target code and files, or "run-time" code.

For older generation SigC641x processor software, the target .rar file should be installed (unzipped) on a WinXP or Win7/8 machine, to the C:\ directory. This allows all project file and subfolder references inside the .rar file to remain intact. CCS3 and CCS5 project files and DSPBIOS and SYSBIOS configuration files (.tcf and .cfg files, respectively) are provided.

For SigC641x modules, the target software folder hierarchy is shown in Figure 4-2b below.

 Copyright © Signalogic 2012-2015

Figure 4-2b, Target software folder hierarchy for SigC641x modules

For SigC641x cards, TI code generation and configuration is not automated and must be performed with TI's Code Composer Studio IDE environment, using project files included in the target software. In this case, the target .rar file has to be installed on a Win machine, as shown in Figure 4-2b above.

4.2 Building the Loadable Kernel Module (LKM) Driver

Note: All driver builds and loads are performed during installation using the automated install script. This section is provided to show how to build drivers manually, if for any reason that should be required.

Files for the host software DirectCore driver reside on the "driver" subfolder under the "DirectCore" subfolder (see Figure 4-1 above). Key files include:

sig_mc_hw.c	SigC667x and SigC641x driver source code
Makefile	Makefile to build and load the driver
sig_mc_hw.h	Driver include file (located under ../DirectCore/include)

The DirectCore unified host driver supports all target hardware with interfaces including PCIe, SRIO, and onchip (SoC). User applications that make DirectCore API calls or incorporate CIM source code generation should contain no user space calls (e.g. ioctl, pread, pwrite, etc) directly to the DirectCore driver; instead, these calls are made by DirectCore libraries via user application API calls. Applications that make user space calls directly to the driver are not supported.

Note 1: To build the DirectCore driver, certain Linux kernel header files are required. These are available if the "kernel-devel" RPM has been installed, and a symlink has been created for "/usr/src/linux". See section 4.2.2 below for more information.

Note 2: In the instructions below, the "#" symbol is used to denote a Linux command-line prompt. It should not be part of any actual command entry.

First go to the driver folder:

```
#cd /install_path/Signalogic/DirectCore/driver
```

To clean the driver folder (optional), enter:

```
#make clean
```

To build the driver, enter:

```
#make
```

To load the driver, enter:

```
#make load
```

 Copyright © Signalogic 2012-2015

After loading the driver, you should see screen output similar to Figure 4-3a below:

```
driver$ make load
sync
insmod sig_mc_hw.ko
chmod 666 /dev/sig_mc_hw
../hw_utils/driver_query

Module: sig_mc_hw
   Number of C6678 devices detected: 16
   Number of C6678 cores detected:   128
   Number of reserved C6678 cores:   0

Chip Status:
   Chip 0: Available
   Chip 1: Available
   Chip 2: Available
   Chip 3: Available
   Chip 4: Available
   Chip 5: Available
   Chip 6: Available
   Chip 7: Available
   Chip 8: Available
   Chip 9: Available
   Chip 10: Available
   Chip 11: Available
   Chip 12: Available
   Chip 13: Available
   Chip 14: Available
   Chip 15: Available
```

4-3a, SigC667x manual driver load screen output

To unload the driver, enter:

```
#make unload
```

Figure 4-3b below is a screen capture showing files on the driver subfolder and the manual driver unload process:

```
driver$ ls -o
total 1264
-rw-r--r-- 1 root   2980 Nov  2 16:45 Makefile
-rw-r----- 1 root     55 Nov  6 17:45 modules.order
-rw-r--r-- 1 root    690 Feb 25  2015 Module.symvers
-rw-r--r-- 1 root 237944 Oct 30 11:53 sig_mc_hw.c
-rw-r--r-- 1 root     99 Nov  6 12:05 sig_mc_hw.modules
driver$ make unload
rmmod sig_mc_hw
driver$
```

 Copyright © Signalogic 2012-2015

Figure 4-4b, SigC641x / SigC667x manual driver unload process

4.2.1 Old Style Driver Notes

If you are using versions of DirectCore software version prior to v5, then you may have the "old style" driver. In this case, after loading the driver, you can perform the following command to verify the driver loaded correctly:

```
#ls /dev/
```

For SigC6678-32 cards, you should see at least four (4) entries of the form:

```
/dev/sigc667x_n
```

where n is 0 to 7. For SigC6678-64 cards you should see at least eight (8) entries, and for SigC641x cards or modules, you should see at least one (1) entry. More entries will be shown if multiple cards are installed.

Figure 4-3b below is a screen capture showing the driver process -- clean, make, and load -- and successful results verified with command:

```
ls /dev/
```

```
[root@genstor driver]# make clean
rm -rf *.ko
rm -rf *.o
rm -rf *.mod.ko
rmmod sig_mc_hw
ERROR: Module sig_mc_hw does not exist in /proc/modules
make: [clean] Error 1 (ignored)
[root@genstor driver]# make
make -C /usr/src/linux/ M=/root/Signalogic_2012v4/DirectCore/driver modules
make[1]: Entering directory `/usr/src/kernels/2.6.32-279.2.1.el6.x86_64'
  CC [M]  /root/Signalogic_2012v4/DirectCore/driver/sig_mc_hw.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/Signalogic_2012v4/DirectCore/driver/sig_mc_hw.mod.o
  LD [M]  /root/Signalogic_2012v4/DirectCore/driver/sig_mc_hw.ko.unsigned
  NO SIGN [M] /root/Signalogic_2012v4/DirectCore/driver/sig_mc_hw.ko
make[1]: Leaving directory `/usr/src/kernels/2.6.32-279.2.1.el6.x86_64'
[root@genstor driver]# make load
sync
insmod sig_mc_hw.ko
[root@genstor driver]# ls /dev/
autofs          disk     hugepages  loop5    mem                pts    ram3    rtc         sigc667x_0
block           fb       hvc0       loop6    net                ram0   ram4    rtc0        sigc667x_1
bsg             fb0      input      loop7    network_latency    ram1   ram5    sda         sigc667x_2
bus             fd       kmsg       lp0      network_throughput ram10  ram6    sda1        sigc667x_3
char            full     log        lp1      null               ram11  ram7    sda2        snapshot
console         fuse     loop0      lp2      nvram              ram12  ram8    sda3        stderr
core            hidraw0  loop1      lp3      oldmem             ram13  ram9    serial      stdin
cpu             hidraw1  loop2      MAKEDEV  port               ram14  random  sg0         stdout
cpu_dma_latency hidraw2  loop3      mapper   ppp                ram15  raw     shm         systty
crash           hpet     loop4      mcelog   ptmx               ram2   root    sigc641x_0  tty
[root@genstor driver]#
```

Figure 4-4a, SigC641x / SigC667x driver build, load, and verify process

In Figure 4-3, areas outlined in red show a SigC6678-32 card installed (4 6678 devices, 32 CIM cores total), and a SigC641x module installed on a PCIe adapter card (8 C641x devices, 8 CIM cores total). In this case, both cards were installed in the same server simultaneously.

> ***Note:*** ***If the ls /dev/ command does not show device entries as noted above, you may need to reboot the system. This is a rare occurrence, but can happen in some systems and/or depending on which firmware revision is loaded on the SigC641x or SigC667x card. For the latter case, see the notes in section 1.2 above, "Installing SigC667x Cards".***

To unload the driver, enter:

```
#rmmod sig_mc_hw
```

Note that when loading and removing the driver, the following system device entries are automatically updated:

```
/dev/sigc641x_n

/dev/sigc667x_n
```

Where n is the device number for SigC667x, and card number for SigC641x.

Figure 4-4b below is a screen capture showing files on the driver subfolder and the driver unload process:

```
[root@genstor driver]# dir
Makefile        sig_mc_hw.c            sig_mc_hw.mod.c  sig_mc_hw.o
modules.order   sig_mc_hw.ko           sig_mc_hw.mod.o
Module.symvers  sig_mc_hw.ko.unsigned
[root@genstor driver]#
[root@genstor driver]# rmmod sig_mc_hw
[root@genstor driver]#
```

Figure 4-4b, SigC641x / SigC667x driver unload process

4.2.2 Linux Kernel-Devel Build Requirement

> ***Note:*** ***All driver builds and loads are performed during installation using the automated install script. This section is provided if for some reason you are building the driver manually.***

When building the DirectCore driver, the "kernel-devel" package, which contains partial Linux source (in particular, certain header files), must be installed. Also, a symlink for "/usr/src/linux" must be created. Instructions for these steps are given below. Note that full kernel source code is not required, as the DirectCore driver is an external driver, not incorporated within the kernel that can be loaded -- and removed -- as needed after the kernel is already booted and running (i.e. loadable kernel module, or LKM).

Note: The commands below assume that you are logged in as root.

1. **Step 1.** If you are running a standard Linux kernel you can install the kernel-devel package by following the commands below.

For CentOS and Fedora:

```
[root@host]# yum install kernel-devel.$(uname -r)
```

For Ubuntu and Debian:

```
[root@host]# apt-get install linux-headers-$(uname -r)
```

Note the use of an embedded "uname –r" command in the above command lines. The purpose of this is to ensure installation of the kernel-devel package that **matches** the currently running kernel.

Note that the "uname –r" command can be used separately to determine the version of the currently running kernel:

```
[root@host]# uname -r
```

The result will look similar to this (depending on what kernel revision is installed):

```
2.6.18-400.1.1.el5xen
```

Here are some examples without using an embedded "uname –r" command. Note that these may or may not have exactly correct entry *(comments in italics)*:

```
[root@host]# yum -v install kernel-devel.i686      (for 32-bit CentOS)
[root@host]# yum -v install kernel-devel.x86_64    (for 64-bit CentOS)
[root@host]# apt-get install linux-headers-x86_64  (for 64-bit Ubuntu)
```

2. **Step 2.** In addition to installing the kernel-devel package, a symlink for "/usr/src/linux" must be created, using a command similar to the following:

```
[root@host]# ln -s /usr/src/linux-xx.xx.xx /usr/src/linux
```

where "xx.xx.xx" matches the kernel version returned by the "uname –r" command shown above.

After the symlink is created, the following ls (list) command can be used to verify the symlink (shown as the last line of the ls command result):

 Copyright © Signalogic 2012-2015

```
[root@localhost bin]# ls -l /usr/src/

    total 8

    drwxr-xr-x. 2 root root 4096 Sep 23  2011 debug

    drwxr-xr-x. 5 root root 4096 Apr  8  2014 kernels

    lrwxrwxrwx. 1 root root   34 Nov 13  2012 linux -> kernels/2.6.32-279.14.1.el6.x86_64
```

After these two steps have been performed, the DirectCore driver makefile can be executed, as described in section 4.1 above.

 Copyright © Signalogic 2012-2015

4.3 Building DirectCore Libraries

***Note:** All library builds are performed during installation using the automated install script. This section is provided to show how to build libraries manually, if for any reason that should be required.*

Source code for DirectCore libraries is located on subfolders under:

```
/install_path/Signalogic/DirectCore/lib/
```

as shown in Figure 4-1, above. If libraries are being built for the first time on a new system, or being rebuilt for a software update on an existing system, the build sequence should be as follows:

1. enmgr	***Note:** Currently these libraries are supported by all hardware types*
2. hwmgr	
3. hwlib	
4. cimlib	
5. filelib	
6. tdmlib	***Note:** Currently these libraries are only supported by SigC641x hardware. Building and/or running these libraries should not be attempted for SigC667x hardware*
7. sessionmgr	
8. voplib	

For each library, after building, either an archive (.a) or shared object (.so) file should be copied to the default system library directory, typically /usr/lib. Note that the Makefiles do this automatically, but it's always a good idea to verify that library files are being copied; otherwise, application layer programs that use DirectCore APIs may not build correctly.

For each library, use the following build sequence:

```
#cd ../lib/<libname>

#make clean

#make
```

Figure 4-5 below shows some of the DirectCore libraries being rebuilt.

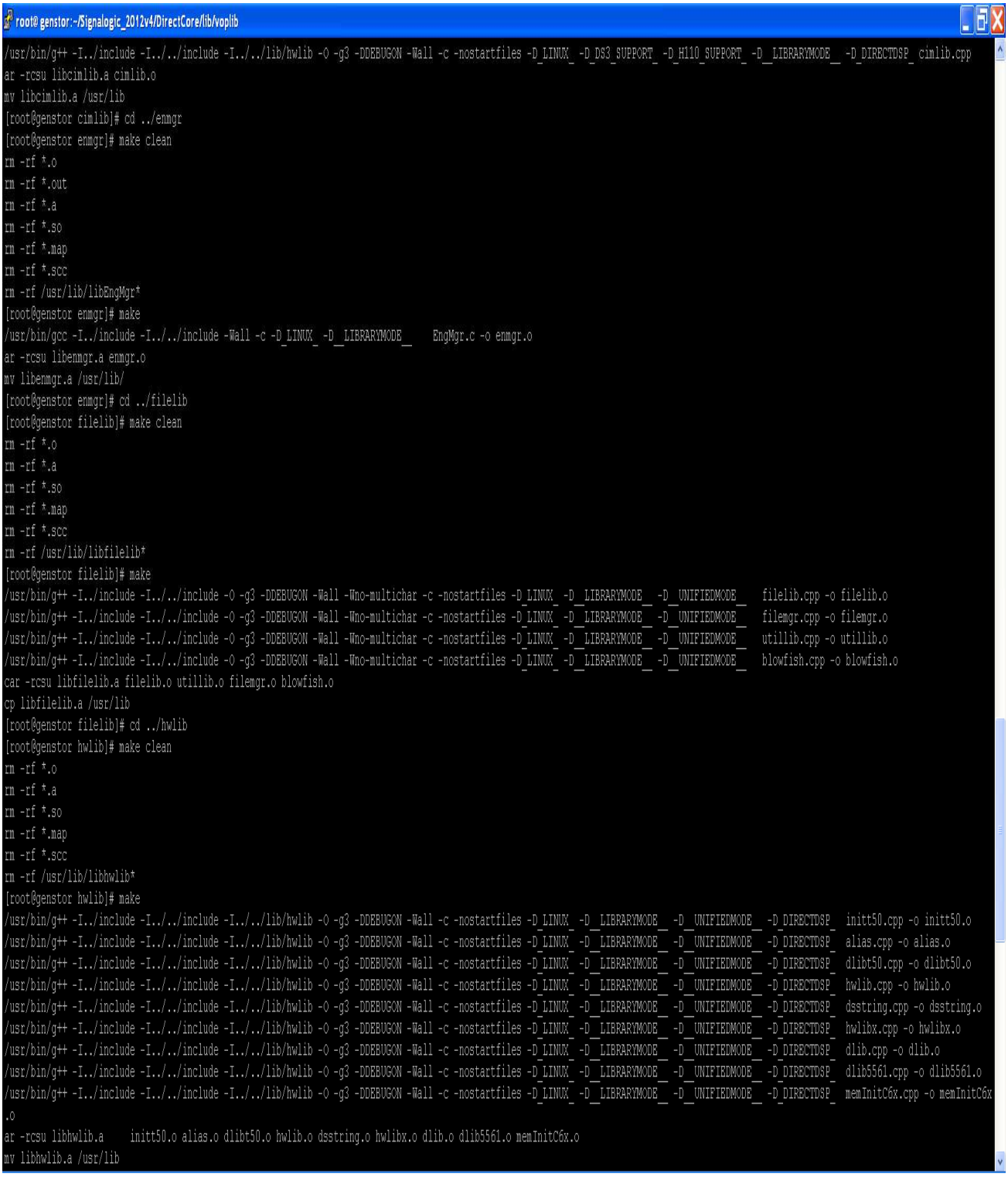

root@genstor:~/Signalogic_2012v4/DirectCore/lib/voplib
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D_DS3_SUPPORT_ -D_H110_SUPPORT_ -D__LIBRARYMODE__ -D_DIRECTDSP_ cimlib.cpp
ar -rcsu libcimlib.a cimlib.o
mv libcimlib.a /usr/lib
[root@genstor cimlib]# cd ../enmgr
[root@genstor enmgr]# make clean
rm -rf *.o
rm -rf *.out
rm -rf *.a
rm -rf *.so
rm -rf *.map
rm -rf *.scc
rm -rf /usr/lib/libEngMgr*
[root@genstor enmgr]# make
/usr/bin/gcc -I../include -I../../include -Wall -c -D_LINUX_ -D__LIBRARYMODE__ EngMgr.c -o enmgr.o
ar -rcsu libenmgr.a enmgr.o
mv libenmgr.a /usr/lib/
[root@genstor enmgr]# cd ../filelib
[root@genstor filelib]# make clean
rm -rf *.o
rm -rf *.a
rm -rf *.so
rm -rf *.map
rm -rf *.scc
rm -rf /usr/lib/libfilelib*
[root@genstor filelib]# make
/usr/bin/g++ -I../include -I../../include -O -g3 -DDEBUGON -Wall -Wno-multichar -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ filelib.cpp -o filelib.o
/usr/bin/g++ -I../include -I../../include -O -g3 -DDEBUGON -Wall -Wno-multichar -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ filemgr.cpp -o filemgr.o
/usr/bin/g++ -I../include -I../../include -O -g3 -DDEBUGON -Wall -Wno-multichar -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ utillib.cpp -o utillib.o
/usr/bin/g++ -I../include -I../../include -O -g3 -DDEBUGON -Wall -Wno-multichar -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ blowfish.cpp -o blowfish.o
car -rcsu libfilelib.a filelib.o utillib.o filemgr.o blowfish.o
cp libfilelib.a /usr/lib
[root@genstor filelib]# cd ../hwlib
[root@genstor hwlib]# make clean
rm -rf *.o
rm -rf *.a
rm -rf *.so
rm -rf *.map
rm -rf *.scc
rm -rf /usr/lib/libhwlib*
[root@genstor hwlib]# make
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ initt50.cpp -o initt50.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ alias.cpp -o alias.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ dlibt50.cpp -o dlibt50.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ hwlib.cpp -o hwlib.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ dsstring.cpp -o dsstring.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ hwlibx.cpp -o hwlibx.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ dlib.cpp -o dlib.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ dlib5561.cpp -o dlib5561.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ memInitC6x.cpp -o memInitC6x
.o
ar -rcsu libhwlib.a initt50.o alias.o dlibt50.o hwlib.o dsstring.o hwlibx.o dlib.o dlib5561.o memInitC6x.o
mv libhwlib.a /usr/lib

Figure 4-5, DirectCore library rebuild process

After building each library, resulting library filenames should be (in `/usr/lib/`):

 Copyright © Signalogic 2012-2015

libenmgr.a	
libhwmgr.a	
libhwlib.so	
libcimlib.so	
libfilelib.a	
libtdmlib.a	***Note:*** ***See notes above about whether to build these libraries.***
libsessionmgr.a	
libvoplib.a	

 Copyright © Signalogic 2012-2015

4.4 Test Programs

The following test programs are available after default software installation.

Test Program	Card Connections Required	Comments
memTest	None: No loopback connections or other hardware connections are required for this tests	Can be used to validate the card and driver installation. No DirectCore APIs are used, and only executable code for purposes of one-time initialization runs on the target. This is the most minimal test available
boardTest	None: No loopback connections or other hardware connections are required for these tests. The demo "simulates" analog input using interrupt driven executable code running on the target, but no actual physical connection is required	Can be used to validate the card, driver, and library installation
fftTest	None	Example of algorithm parallelization and acceleration
streamTest	Ethernet	Example of high performance video transcoding and streaming. UDP/RTP streaming output can be routed to a mobile device, for example an Android tablet running MPlayer or VideoLAN VLC players
vid_streaming conv	None	CIM® software demos, showing example OpenMP superset pragmas in C source code, source code stream separation and generation, and run-time execution on the target hardware (note -- this example is covered in sections 4.7.2 and 4.7.3, vid_streaming and convolution demos)
appTest	Ethernet	***Only supports SigC641x hardware.*** Can be used to set up voice and video connections and perform testing and measurements. The minimum required for appTest is Ethernet (IP) interfaces between the host server and at least one other test server or PC

4.4.1 memTest

***Note:** this section assumes you have already built and loaded the DirectCore target multicore CPU driver, as explained in section 4.1 above, Building the Loadable Kernel Module (LKM) Driver.*

The purpose of memTest is to test hardware and driver software directly and at the lowest possible level, with no libraries and minimal user space code. This is not the normal or recommended method of using SigC641x modules and SigC667x cards but serves instead as a diagnostic and measurement tool. The memTest program has several test modes as follows:

- diagnostic test
- random block test
- broadcast test
- performance test

To build memTest, enter:

```
#cd /install_path/Signalogic/DirectCore/apps/SigC641x_C667x/memTest

#make
```

To run memTest, first enter:

```
#./memTest -h
```

This will bring up the help menu showing command line options that can be used to configure the test program and SigC641x or SigC667x hardware.

memTest command line options are shown below ('+' indicates mandatory entry):

-c<*string*>	+	Card designator, e.g. -c*SIGC66XX* or -c*SIGC64XX*
-B<*num*>		Block size
-d		Diagnostic test (default if -*d*, -*r*, or -*z* not entered)
-D		Direct mode (access card registers directly)
-m<*num*>	+	Bitmask for core list (1 = core0, 3 = cores 0 and 1, etc)

-M<*string*>	Memory region for SigC667x, enter *-Moc*, *-Mos*, or *-Me* (onchip mem, onchip shared mem, or external mem)
-n<*num*>	Number of loops (otherwise runs until keyboard exit)
-p<*char*>	Performance measurement, enter *-p* or *-pt* (*-pt* measures data transfer time only)
-P	Master push mode (wait for transfer to complete)
-r	Random address block test
-R	Read only
-W	Write only
-z	Broadcast test

<num> entries may be specified in either decimal or hex format; if hex, then use a '0x' prefix.

Note that -d, -r, and -z are the possible test types; if none are entered, then the default test type is diagnostic (-d). If more than one test type is entered, then tests are run in the following sequence:

1. diagnostic
2. broadcast
3. random block

Note that diagnostic test runs until a key is pressed, so if multiple test types are entered including diagnostic, then you must exit the diagnostic test (press a key) in order to complete all tests.

Here are sample command lines to run memTest for cores 0 and 1, using the diagnostic test type:

```
#./memTest -m3 -d -cSIGC66XX

#./memTest -m3 -d -cSIGC64XX
```

4.4.1.1 memTest Output Description

Figure 4-6 below is a screen capture of a successful memTest run for 3 cores by running the following command:

 Copyright © Signalogic 2012-2015

```
#./memTest -m7 -d -cSIGC66XX
```

or

```
#./memTest -m7 -d -cSIGC64XX
```

The screen capture in Figure 4-6 below shows an incrementing loop counter at the top of the terminal display and the error counters for the first 3 cores remain zero as the loop counter value increases. The remaining error counters represent cores not included in the "-m" "core list" parameter given in the command line. These are shown as "n/a" to indicate they are not subject to the ongoing test.

To exit from the program, press any key.

> ***Note!*** ***For SigC667x hardware, "core list" with multiple bits set may span one or more target multicore CPUs. For SigC641x hardware, "core list" has the same meaning as "processor list", as the C641x series of devices is single-core, and thus any core list value with more than one bit set specifies more than one device (CPU).***

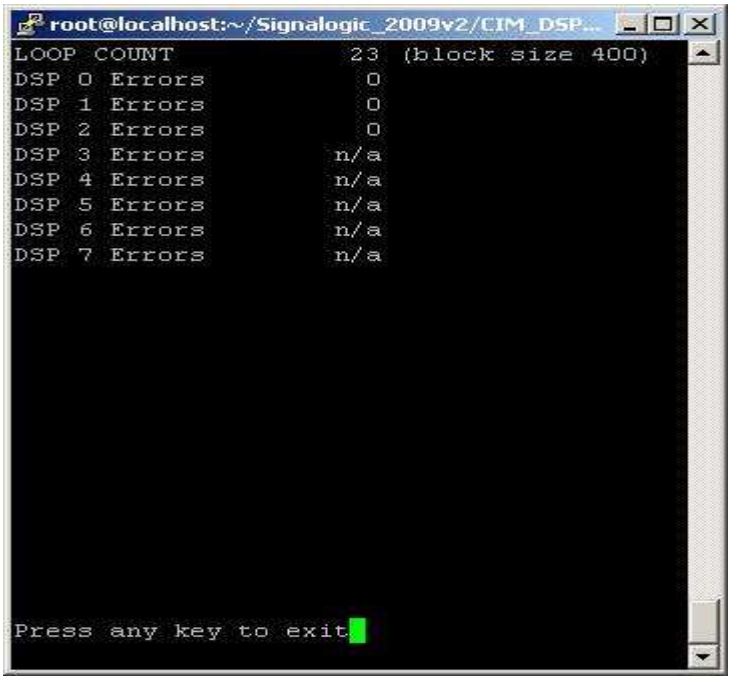

root@localhost:~/Signalogic_2009v2/CIM_DSP...
LOOP COUNT 23 (block size 400)
DSP 0 Errors 0
DSP 1 Errors 0
DSP 2 Errors 0
DSP 3 Errors n/a
DSP 4 Errors n/a
DSP 5 Errors n/a
DSP 6 Errors n/a
DSP 7 Errors n/a
Press any key to exit

Figure 4-6, memTest running diagnostic test for 3 cores on SigC667x or SigC641x hardware

Figure 4-7 below is a screen capture of a memTest diagnostic run for 32 cores, using SigC667x hardware, by executing the following command:

```
#./memTest -cSIGC66XX -d -m0xffffffff -B98304 -P
```

 Copyright © Signalogic 2012-2015

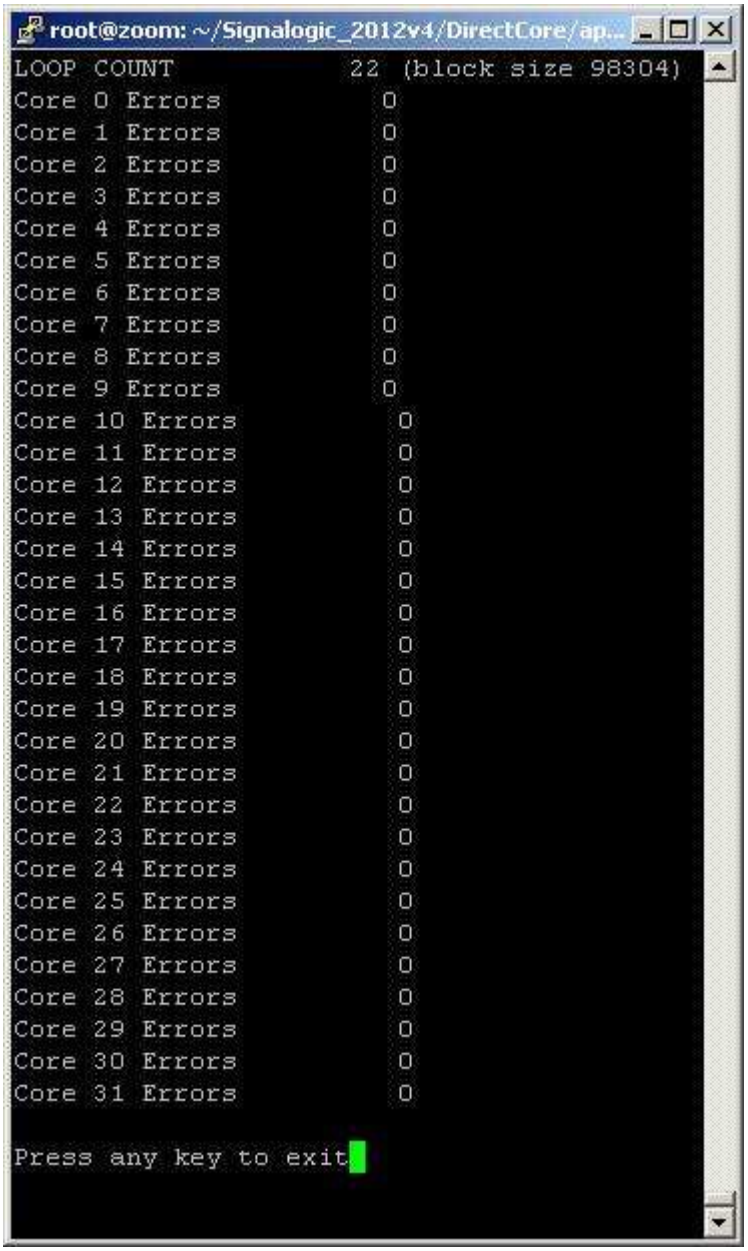

root@zoom: ~/Signalogic_2012v4/DirectCore/ap...
LOOP COUNT 22 (block size 98304)
Core 0 Errors 0
Core 1 Errors 0
Core 2 Errors 0
Core 3 Errors 0
Core 4 Errors 0
Core 5 Errors 0
Core 6 Errors 0
Core 7 Errors 0
Core 8 Errors 0
Core 9 Errors 0
Core 10 Errors 0
Core 11 Errors 0
Core 12 Errors 0
Core 13 Errors 0
Core 14 Errors 0
Core 15 Errors 0
Core 16 Errors 0
Core 17 Errors 0
Core 18 Errors 0
Core 19 Errors 0
Core 20 Errors 0
Core 21 Errors 0
Core 22 Errors 0
Core 23 Errors 0
Core 24 Errors 0
Core 25 Errors 0
Core 26 Errors 0
Core 27 Errors 0
Core 28 Errors 0
Core 29 Errors 0
Core 30 Errors 0
Core 31 Errors 0
Press any key to exit

Figure 4-7, memTest running diagnostic test for 32 cores on a SigC667x card

Figure 4-8 below is a screen capture of a successful random block memTest run for 3 cores, using SigC641x hardware, by executing the following command:

```
#./memTest -m0xa8 -r-B204 -n20 -cSIGC64XX
```

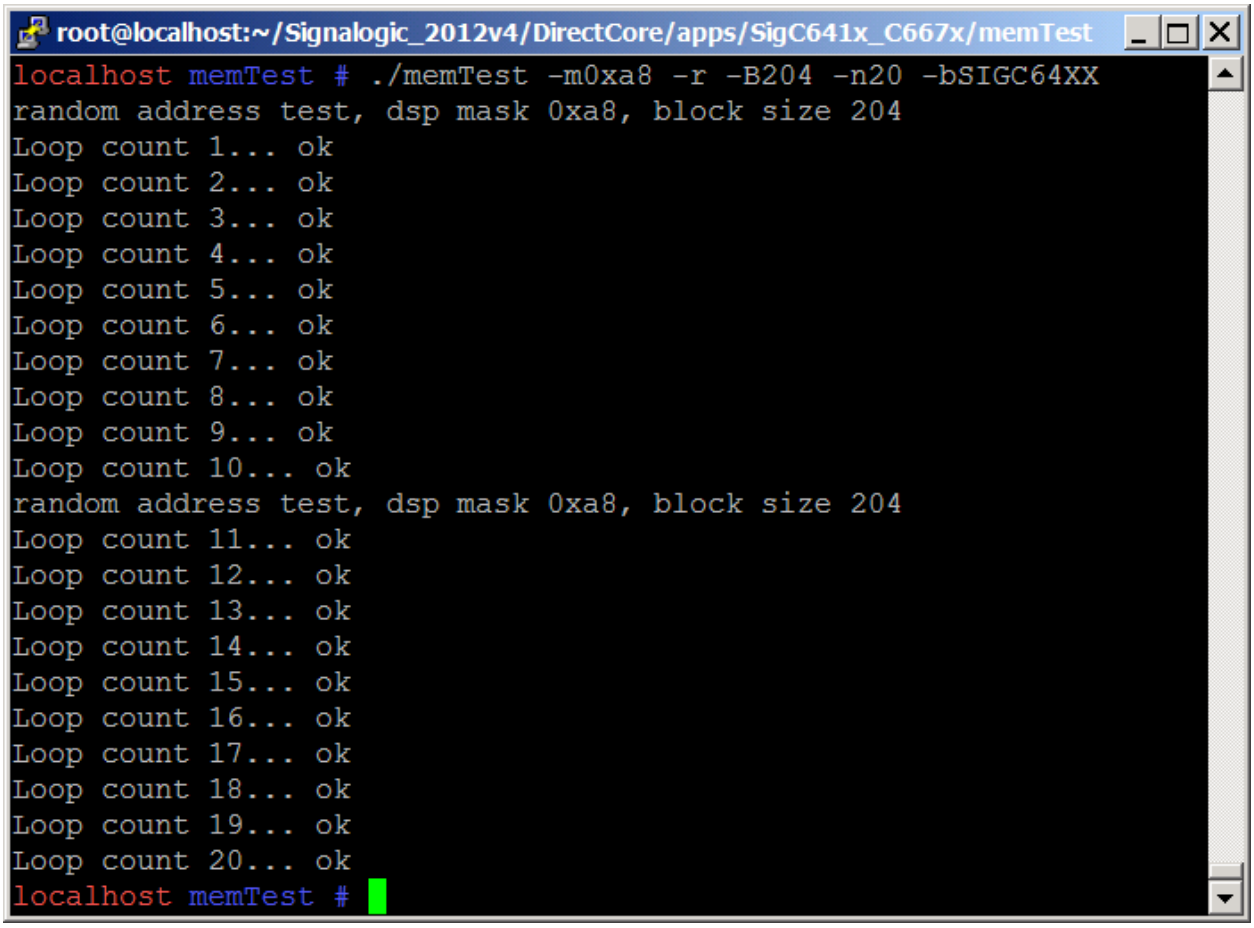

root@localhost:~/Signalogic_2012v4/DirectCore/apps/SigC641x_C667x/memTest
localhost memTest # ./memTest -m0xa8 -r -B204 -n20 -bSIGC64XX
random address test, dsp mask 0xa8, block size 204
Loop count 1... ok
Loop count 2... ok
Loop count 3... ok
Loop count 4... ok
Loop count 5... ok
Loop count 6... ok
Loop count 7... ok
Loop count 8... ok
Loop count 9... ok
Loop count 10... ok
random address test, dsp mask 0xa8, block size 204
Loop count 11... ok
Loop count 12... ok
Loop count 13... ok
Loop count 14... ok
Loop count 15... ok
Loop count 16... ok
Loop count 17... ok
Loop count 18... ok
Loop count 19... ok
Loop count 20... ok
localhost memTest #

Figure 4-8, memTest running random block test for 3 cores on a SigC641x module

Note that in the above command, the `-n20` parameter specifies the test to run for 20 iterations. Without the `-n` parameter the test runs for one iteration by default.

Figure 4-9 below is a screen capture of a successful random block, write-only, test of onchip core memory (i.e. core-specific L2 memory) for 32 cores by executing the following command:

```
#./memTest -cSIGC66XX -B98304 -pt -P -r -m0xffffffff -Moc -n10 -W
```

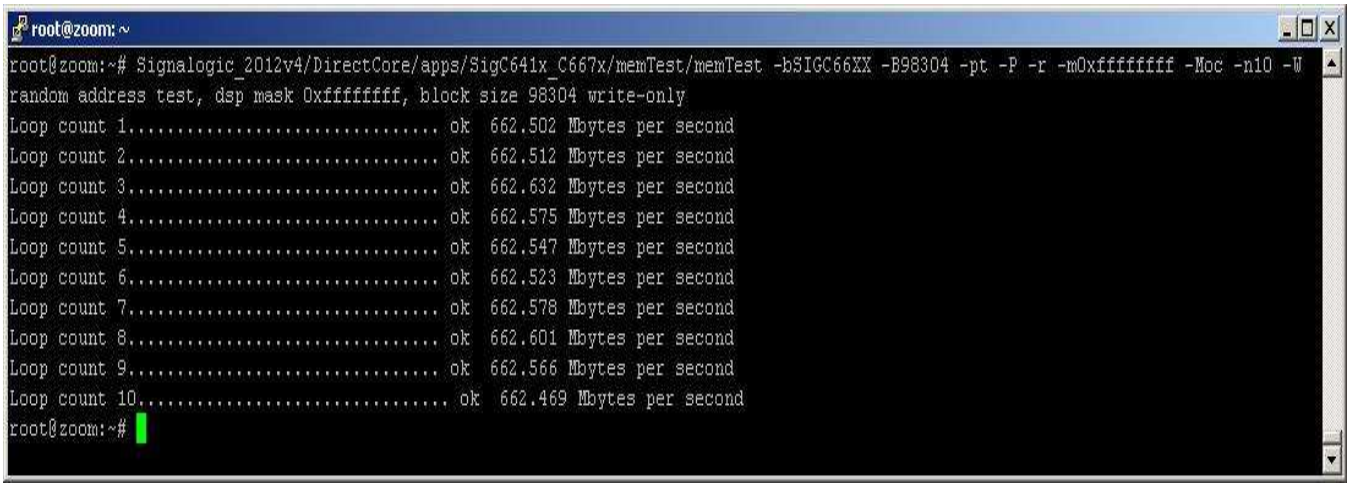

root@zoom: ~
root@zoom:~# Signalogic_2012v4/DirectCore/apps/SigC641x_C667x/memTest/memTest -bSIGC66XX -B98304 -pt -P -r -m0xffffffff -Moc -n10 -W
random address test, dsp mask 0xffffffff, block size 98304 write-only
Loop count 1..... ok 662.502 Mbytes per second
Loop count 2..... ok 662.512 Mbytes per second
Loop count 3..... ok 662.632 Mbytes per second
Loop count 4..... ok 662.575 Mbytes per second
Loop count 5..... ok 662.547 Mbytes per second
Loop count 6..... ok 662.523 Mbytes per second
Loop count 7..... ok 662.578 Mbytes per second
Loop count 8..... ok 662.601 Mbytes per second
Loop count 9..... ok 662.566 Mbytes per second
Loop count 10..... ok 662.469 Mbytes per second
root@zoom:~#

Figure 4-9, memTest running random block test of onchip core-specific memory for 32 cores, with performance measurement enabled

Note that in the above command, the `-W` parameter indicates "write only". Also `-R` can be entered for read-only. If neither is entered, then the test assumes both read and write.

Also note in the above command that "performance measurement" has been specified using the --p option; in this case, specifically, the –pt option, which measures "transfer time only", excluding any intermediate software functions such as data preparation, error checking, etc.

Figure 4-10 below is a screen capture of a successful random block test of onchip shared memory (i.e. MCSM, or multicore shared memory) for 32 cores, using SigC667x hardware, by executing the following command:

```
#./memTest -cSIGC66XX -B0x100000 -pt -P -r -m0xffffffff -Mos -n10
```

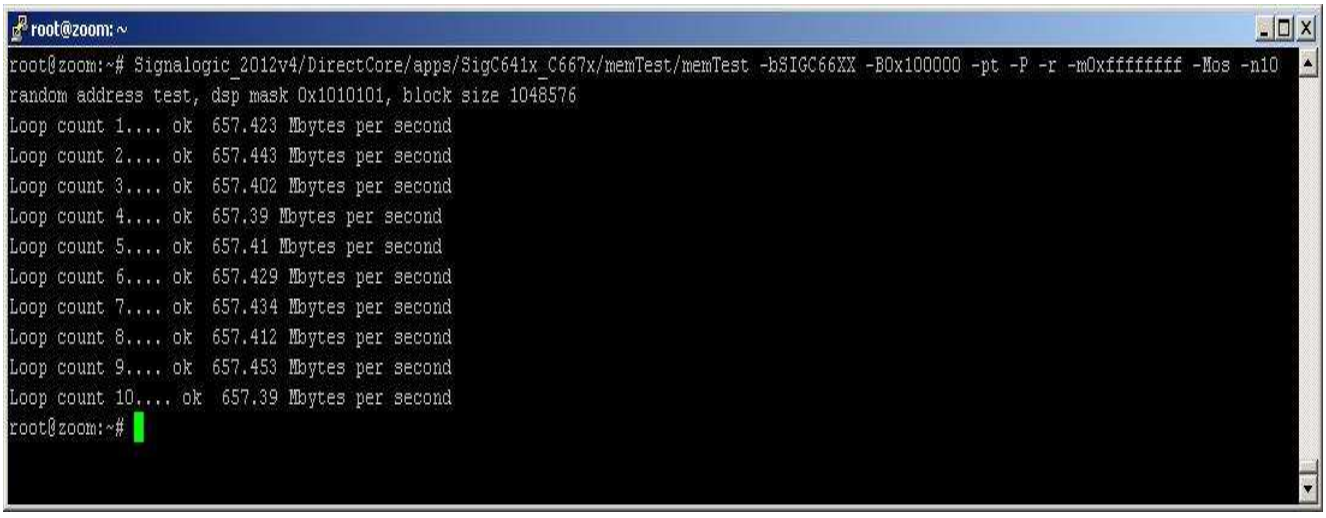

root@zoom: ~
root@zoom:~# Signalogic_2012v4/DirectCore/apps/SigC641x_C667x/memTest/memTest -bSIGC66XX -B0x100000 -pt -P -r -m0xffffffff -Mos -n10
random address test, dsp mask 0x1010101, block size 1048576
Loop count 1.... ok 657.423 Mbytes per second
Loop count 2.... ok 657.443 Mbytes per second
Loop count 3.... ok 657.402 Mbytes per second
Loop count 4.... ok 657.39 Mbytes per second
Loop count 5.... ok 657.41 Mbytes per second
Loop count 6.... ok 657.429 Mbytes per second
Loop count 7.... ok 657.434 Mbytes per second
Loop count 8.... ok 657.412 Mbytes per second
Loop count 9.... ok 657.453 Mbytes per second
Loop count 10.... ok 657.39 Mbytes per second
root@zoom:~#

Figure 4-10, memTest running random block test of onchip shared memory for 32 cores, with performance measurement enabled

 Copyright © Signalogic 2012-2015

Note in the above command, if the memory area specified by the memTest command is shared between all cores of a multicore CPU or target device (e.g. MCSM memory or external DDR3 SDRAM memory), then the memory area is actually tested only once. In this case, memTest will report test results only for the first core of multiple cores per device, which can be seen in Figure 4-10 above by examining the "mask" value printed along with test results.

4.4.1.2 Block Size Parameter Notes

1. For SigC641x, 4 kByte is the maximum block size for all memory areas. For SigC667x, 1 MByte is the maximum block size for external memory and shared memory (MCSM), and 96 kByte is the maximum size for core-specific L2 memory.

2. Up to a point, larger block sizes will increase measured performance; however, improvement is marginal beyond some point. Smaller block sizes are likely to decrease performance. Using SigC667x as an example, from 1 MByte down to 100 KByte block size, performance may decrease about 40 MByte/sec, at 50 KByte block size performance may decrease another 35 MByte/sec, and at 10 KByte block size performance may decrease as much as 200 MByte/sec and be as low as 350 MByte/sec.

3. If a SigC641x module is being used with a PCIe adapter containing the IDT (formerly Tundra Semiconductor) TSi384 PCIe-toPCI bridge chip, then a carefully selected block size should be specified in memTest commands, for example:

```
#./memTest –m0xa8 –r –B204 –n20 –cSIGC64XX
```

The `–B204` parameter specifies the block size at 204 bytes. This is the maximum block size for burst reads/writes that can be used without errors occurring when using a PCIe adapter with the Tundra chip. Specifying this block size compensates for an apparent limitation on PCIe transfer throughput with the TSi384 bridge chip.

4.4.2 boardTest

***Note:** this section assumes (i) the DirectCore target multicore CPU driver has been built and is currently loaded (either during the automated install process, or manually as explained in section 4.1 above, Building the Loadable Kernel Module (LKM) Driver), and (ii) DirectCore libraries have been built (either during the automated install process, or manually, as explained in section 4.2 above, Building DirectCore Libraries).*

The boardTest program gives an example of communicating with target code as it runs in real-time, demonstrating DirectCore library API calls.

To build the boardTest program, enter the following:

```
#cd /install_path/Signalogic/DirectCore/apps/SigC641x_C677x/boardTest

#make
```

Before running boardTest, first enter:

```
#./boardTest -h
```

This will bring up the help menu in order to configure SigC667x or SigC641x hardware.

Command line option syntax (options with "+" sign are mandatory):

-f<*num*>	+	Device clock rate in MHz (e.g. -*f1000*)
-m<*num*>	+	Core select bit mask (e.g. -*m1*, means core0, -*m2* means core1). Entry can be in hex, for example -m0x3f, which means run using 6 cores
-e<*string*>	+	Target executable file, for example -*ebdtest.out*. (ELF and COFF formats are supported for target executable files)
-c<*string*>	+	Card designator, for example -*cSIGC66XX*-8, which indicates that 8 cores should be allocated, or -*cSIGC66XX-32*, which indicates that 32 cores should be allcoated
-T		Run with talker enabled
-v		Run in verbose mode
-h		Display this help list

 Copyright © Signalogic 2012-2015

Here are some example boardTest command lines:

```
# ./boardTest -m1 –f1000 -ebdtest.out -cSIGC64XX-8
```

or

```
# ./boardTest -m1 –f1250 -ebdtest.out -cSIGC66XX-8
```

Note that in the above command-line, the "`-ebdtest.out`" option specifies a target CPU executable code file (in ELF or COFF format). The clock rate will be 1 GHz or 1.25 GHz as specified by the `–f1000` or `–f1250` command line parameters (note that the `–f` command line parameter uses MHz as its units).

4.4.2.1 boardTest Output Description

Figure 4-11 below is a screen capture of output from the boardTest test program, executed using the following command for a SigC667x PCIe card:

```
# ./boardTest -m1 –f1250 -ebdtest.out -cSIGC66XX
```

or the following command for a SigC641x PCIe card or PTMC module:

```
# ./boardTest -m1 –f1000 -ebdtest.out -cSIGC64XX
```

The screen capture in Figure 4-11 shows a sequence of low frequency sinusoidal signal values. The values are read by the boardTest program from dual real-time data buffers in target memory and displayed repeatedly (about 1 sec rate), and will change in each read cycle. Figure 4-11 shows about four (4) of the read cycles -- note the changing data pattern.

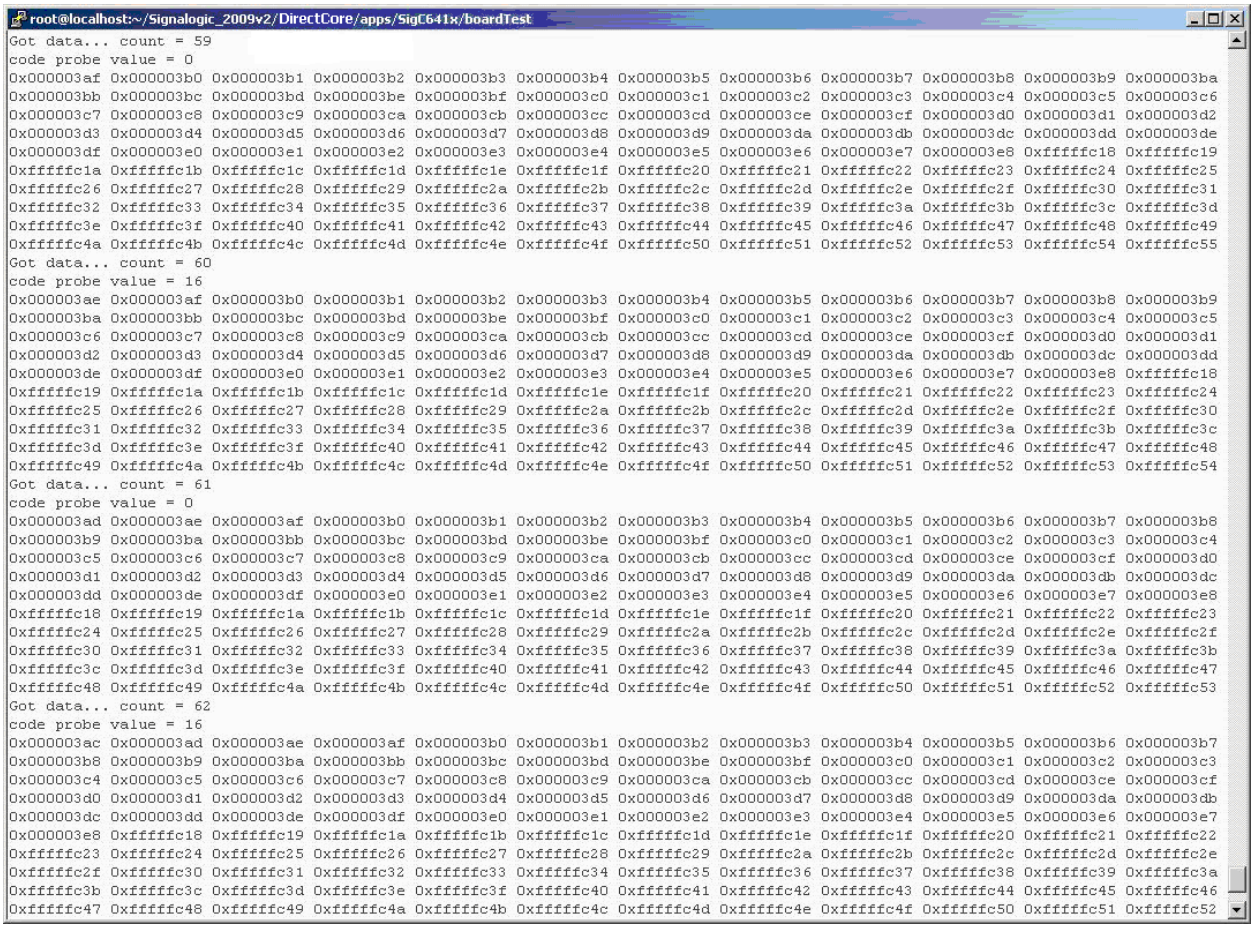


Figure 4-11, Example boardTest display

To exit from the boardTest program, press the q (Quit) or ESC keys.

> ***Note:*** ***you can also exit using Ctrl-C, but using the interactive keyboard commands is preferred as that allows the test program to free (de-allocate) the card handle, close its instance of the driver, and perform other clean up operations. This helps to maintain "clean" host Linux / server operation, and avoid memory leaks, driver errors, etc***

4.4.2.2 C++ Target

It's also possible to build boardTest by specifying a C++ target, which causes a different source code file to be used during the make. The C++ source code uses a "target class" to reduce the number of library calls and somewhat simplifies the amount of code necessary to set up basic target card / module connectivity and data transfer. To build with the C++ target, enter the following *(comments in italics)*:

```
#make cpp    (uses C++ source code)
```

 Copyright © Signalogic 2012-2015

Currently, the "straight make" (no target specified) is the standard (default) option, and should always be tested first.

Note 1: ***The C++ class is currently only supported for SigC641x hardware.***

Note 2: ***The cpp target is not well maintained and may give some build warnings or errors. Please contact Signalogic if you will need to write application programs using C++***

4.4.3 fftTest

Note 1: ***This section assumes you have already (i) built and loaded the DirectCore target multicore CPU driver, as explained in section 4.1 above, Building the Loadable Kernel Module (LKM) Driver, and(ii) built all DirectCore libraries, as explained in section 4.2 above, Building DirectCore Libraries.***

Note 2: ***Currently fftTest is only supported by SigC667x hardware.***

The fftTest program gives an example of running a parallelized algorithm on multiple target cores, using DirectCore library API calls.

To build the fftTest program, enter the following:

```
#cd /install_path/Signalogic/DirectCore/apps/SigC641x_C677x/fftTest

#make
```

Before running fftTest, first enter:

```
#./fftTest -h
```

This will bring up the help menu in order to configure SigC667x hardware.

Command line option syntax (options with "+" sign are mandatory):

-f<num>	+	Device clock rate in MHz (e.g. -f1000)
-m<num>	+	Core select bit mask (e.g. -m1, means core0, -m2 means core1.(Note: in this test, only one bitmask values of 2^n-1 can be selected, specifying the number of cores to run in parallel)
-e<string>	+	Target executable file (in ELF or COFF format, e.g. -efft.out)
-c<string>	+	Card designator (e.g. -cSIGC66XX)
-a<num>		Algorithm flag, 0 = parallel FFT, 1 = serial FFT (default is parallel)
-n<num>		FFT order, from 2 to 20 (default = 6)
-I<num>		Input waveform type, 0=ramp, 1 = Dirac Delta pulse

-i<string>	Input waveform file (in .wav or .tim format)
-v	Run in verbose mode
-h	Display this help list

Here are some example fftTest command lines:

```
# ./fftTest -m1 -f1250 -efft.out -cSIGC66XX -
```

Note that in the above command-line, the "-efft.out" option specifies a target CPU executable code file (in ELF or COFF format). The clock rate will be 1 GHz or 1.25 GHz as specified by the -f1000 or -f1250 command line parameters (note that the -f command line parameter uses MHz as its units). Parameters -a, -n, -I, and -i can be used to specify algorithm and input options.

4.4.3.1 fftTest Output Description

Figure 4-12 below is a screen capture of output from the fftTest test program, executed using the following command for a SigC667x PCIe card:

```
# ./fftTest -m1 -f1000 -efft.out -cSIGC66XX -n4
```

The screen capture in Figure 4-12 shows output for a small FFT order using a parallelized method based on Bailey's "Four Step" algorithm [1], which "factors" an FFT of order N_o into an NxM matrix of FFTs, where N and M are as close as possible to $\sqrt{N_o}$. FFTs in the matrix are then run on multiple target CPU cores concurrently.

A small FFT order is used for the test to allow results to conveniently viewed, and the test to be repeated in serial (non-parallelized) FFT mode in order to compare values for correctness.

[1] FFTs in External or Hierarchical Memory, David H. Bailey, 1989. Journal of Supercomputing, vol. 4 no. 1 Mar 1990. http://www.davidhbailey.com/dhbpapers/fftq.pdf

 Copyright © Signalogic 2012-2015

```
fftTest$ ./fftTest -m1 -f1000 -offt.out -cSIGC66XX -n4
DirectCore / CIM FFT test program for C64x / C66x hardware
userSpecified = {clock = 1000 MHz, Target Executable = fft.out, Card Type = SIGC66XX, Core List = 1, Algorithm Flag = -1}
CardStr = SIGC66XX
IObaseAddr = 0
Core clock rate (Hz) = 1000000000
Target executable file = fft.out
Algorithm mode flag -1
FFT mode: Parallel
Input type 0
FFT Order is 4
Initialising SigC667x card...
Initialising SigC667x cores with core list 0x1...
DSInitProcessor: qwCoreList = 0x1
SlotIndx = 0
Board_Type 0x8106
Talker not used
DSXInitCard SigC66x
ds66xInitCard:
InitBoardT50 success
DSXInitCard Successful!
Resetting specified devices and cores...
DSXResetCard
DSXResetCard Successful!
Downloading run-time code to specified cores...
Entry point = 0xc000000
test mode addr = 0x802094
Setting multicore CPU Properties...
dwCoreId_addr = 8020a4
writing sin/cos table data, addr = 0xc00d760, M=4, N=4
offset value = 32
Sincos Table
sin[0] = 1.000000        cos[0] = -0.000000
sin[1] = 0.000000        cos[1] = -1.000000
sin[2] = -1.000000       cos[2] = -0.000000
sin[3] = -0.000000       cos[3] = 1.000000
Sincos Table 2
sin[0] = 1.000000        cos[0] = -0.000000
sin[1] = 0.000000        cos[1] = -1.000000
sin[2] = -1.000000       cos[2] = -0.000000
sin[3] = -0.000000       cos[3] = 1.000000
writing test data, RI base addr = 0xc100080
Starting real-time multicore code...
DSXRunCard Successful!
Hardware Running....
inp[0]=0.00 0.00j       1.00 0.00j      2.00 0.00j      3.00 0.00j
inp[4]=4.00 0.00j       5.00 0.00j      6.00 0.00j      7.00 0.00j
inp[8]=8.00 0.00j       9.00 0.00j      10.00 0.00j     11.00 0.00j
inp[12]=12.00 0.00j     13.00 0.00j     14.00 0.00j     15.00 0.00j
Got data... count = 0
fft results:
out data[0] = 120.00 0.00j
out data[1] = -8.00 40.22j
out data[2] = -8.00 19.31j
out data[3] = -8.00 11.97j
out data[4] = -8.00 8.00j
out data[5] = -8.00 5.35j
out data[6] = -8.00 3.31j
out data[7] = -8.00 1.59j
out data[8] = -8.00 0.00j
out data[9] = -8.00 -1.59j
out data[10] = -8.00 -3.31j
out data[11] = -8.00 -5.35j
out data[12] = -8.00 -8.00j
out data[13] = -8.00 -11.97j
out data[14] = -8.00 -19.31j
out data[15] = -8.00 -40.22j
accelerator card runtime: 0.000008 sec
```

Figure 4-12, Example fftTest display output

4.4.4 streamTest

Note 1: ***This section assumes you have already (i) built and loaded the DirectCore target multicore CPU driver, as explained in section 4.1 above, Building the Loadable Kernel Module (LKM) Driver, and(ii) built all DirectCore libraries, as explained in section 4.2 above, Building DirectCore Libraries.***

Note 2: ***Currently streamTest is supported on SigC667x hardware and not supported on SigC641x hardware.***

The streamTest program gives an example of running a video codec algorithm on multiple target cores, using DirectCore library API calls.

To build the streamTest program, enter the following:

```
#cd /install_path/Signalogic/DirectCore/apps/SigC641x_C677x/streamTest

#make
```

Before running streamTest, first enter:

```
#./streamTest -h
```

This will bring up the help menu in order to configure SigC667x hardware.

Command line option syntax (options with "+" sign are mandatory):

```
-f<num>          +    Device clock rate in MHz (e.g. -f1000)

-m<num>          +    Core select list (e.g. -m1, = core0, -m2 =
                      core1, -m3 = cores 1 and 2.(Note:  in this
                      test, only one core can be selected at a
                      time)

-e<string>       +    Target executable file (in ELF or COFF
                      format, e.g. -evideo.out)

-c<string>       +    Card designator (e.g. -cSIGC66XX)

-i<string>            Input unencoded video file (in .yuv
                      format)

-o<string>            Output encoded video file (in .h264
                      format)

-x<num>               X dimension resolution
```

```
-y<num>          Y dimension resolution

-r<num>          Frame Rate (in frames per sec, default is
                 30 fps)

-B<num>          Bitrate of encoded (compressed) output
                 file (default is 800 kbps)

-s<num>      +   Mode, -s0 = file, -s2 = streaming

-v               Run in verbose mode

-h               Display this help list
```

Here are some example streamTest command lines:

```
#./streamTest -m1 -f1250 -s0 -estream.out -cSIGC66XX-8
-i/install_path/Signalogic/video_files/airshow_352x288p_30fps_420fmt.yuv
-x352 -y288 -r30 -B1500000 -oencoded.h264
```

and

```
#./streamTest -m0x03 -f1250 -s2 -estream.out -cSIC66XX-8
-i/install_path/Signalogic/video_files/parkrun_720p_50fps_420fmt.yuv
-x1280 -y720 -r50 -B1500000 -oencoded.h264
```

Note that in the above command lines, the "`-estream.out`" option specifies a target CPU executable code file. Target CPU clock rates can range from 1 to 1.6 GHz as specified by the `-f` command line parameter (e.g. `-f1000, -f1250, -f1600`, etc. Note that the `-f` command line parameter uses MHz as its units). Parameters -i, -o, -x, -y, -r, and -s can be used to specify input and output files, resolution, frame rate, and mode. In `-s0` mode, streamTest operates in "one shot" file-to-file mode. In `-s2` mode, streamTest operates in continuous file-to-file or file-to-stream mode, where a stream is continuous network IP/UDP/RTP packet output.

Note: The High Performance Multicore Network (HPMN) stack software must be loaded on the target accelerator to connect and send RTP output streams (this software requires a license and may not be included in some eval/demo software packages).

4.4.4.1 streamTest Output Description

Figure 4-13 below is a screen capture of output from the streamTest program, executed using the following command for a SigC667x PCIe card:

```
  #./streamTest -m1 -s0 -f1250 -estream.out -cSIGC66XX
-itouchdown_pass_1080p_30fps_420fmt.yuv -x1920 -y1080 -r30 -oencoded.h264
```

The screen capture in Figure 4-13 below shows typical streamTest run-time output, and Figure 4-14 below shows the resulting H.264 encoded output file viewed using the VideoLAN VLC player.

```
videoTest$ ./videoTest -m1 -f1000 -evideo.out -cSIGC66XX -itouchdown_pass_1080p_tr.yuv
DirectCore H.264 algorithm test program for C64x / C66x hardware
userSpecified = {clock = 1000 MHz, Target Executable = video.out, Card Type = SIGC66XX, Core List = 1, Algorithm Flag = -1}
CardStr = SIGC66XX
Core clock rate (Hz) = 1000000000
Target executable file = video.out
Initializing  SigC667x card...
Initializing SigC667x cores with core list 0x1...
DSInitProcessor: qwCoreList = 0x1
SlotIndx = 0
Board_Type 0x8106
Talker not used
DSXInitCard SigC66x
ds66xInitCard:
InitBoardT50 success
DSXInitCard Successful!
Resetting specified devices and cores...
DSXResetCard
DSXResetCard Successful!
Downloading run-time code to specified cores...
call to DSLoadFileCores Success
loading file: touchdown_pass_1080p_tr.yuv
DSLoadDataFile loaded 653184000 bytes from input video file touchdown_pass_1080p_tr.yuv
Entry point = 0xc000000
testrun symbol address = 0x85ce34
errorCode symbol address = 0x857310
numFramesEncoded symbol address = 0x857304
Host test run = 0
Target error code = 0
test mode addr = 0x802094
Setting multicore CPU Properties...
Starting real-time multicore code...
DSXRunCard Successful!
Hardware Running....
Timer running...
Num frames encoded:     210
C66x video encoding complete
Target test run = 101
Target error code = 0
Total number of frames encoded = 210
Number of output bytes85c478
Saving 9041343 bytes to output file encoded.h264
DSXResetCard
DSXResetCard Successful!

Stopped Hardware...
```

Figure 4-13, Example streamTest display output

As a progress update, streamTest will continuously update the current frame (after the line "Num frames encoded" in Figure 4-13 above). If streamTest hangs for any reason, you can exit by pressing the q (Quit) or ESC keys.

***Note:** you can also use Ctrl-C to exit test programs, but the interactive keyboard commands are preferred, as they allow the test program to free (de-allocate) the card handle, close its instance of the driver, and perform other clean up operations. This helps maintain "clean" host Linux operation, and avoid memory leaks, driver errors, etc.*

Figure 4-14 below shows playback of the resulting "encoded.h264" file using the VideoLAN VLC or MPlayer video players (other widely used video players can also be used, such as Windows Media Player).

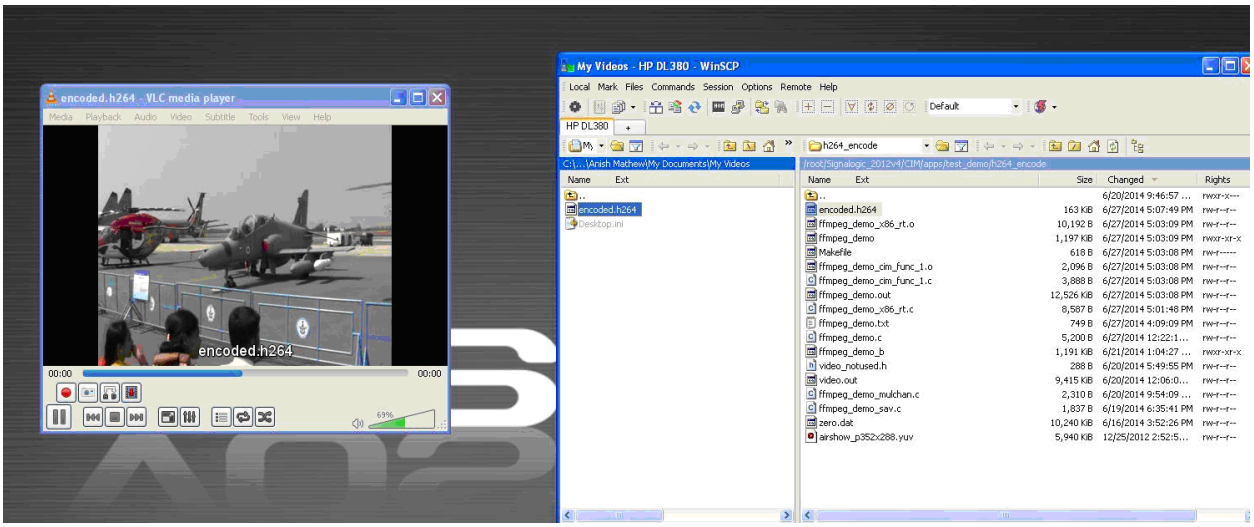


Figure 4-14, Example of encoded streamTest output. In the above screen capture, H.264 encoded data has been stored to a file, and then played back for test / comparison purposes using the VideoLAN VLC open source video player

Figure 4-15 below shows playback on a Ubuntu X11 desktop of the original .yuv file using ffplay, which is a component of ffmpeg. ffplay is available for Linux and Windows.

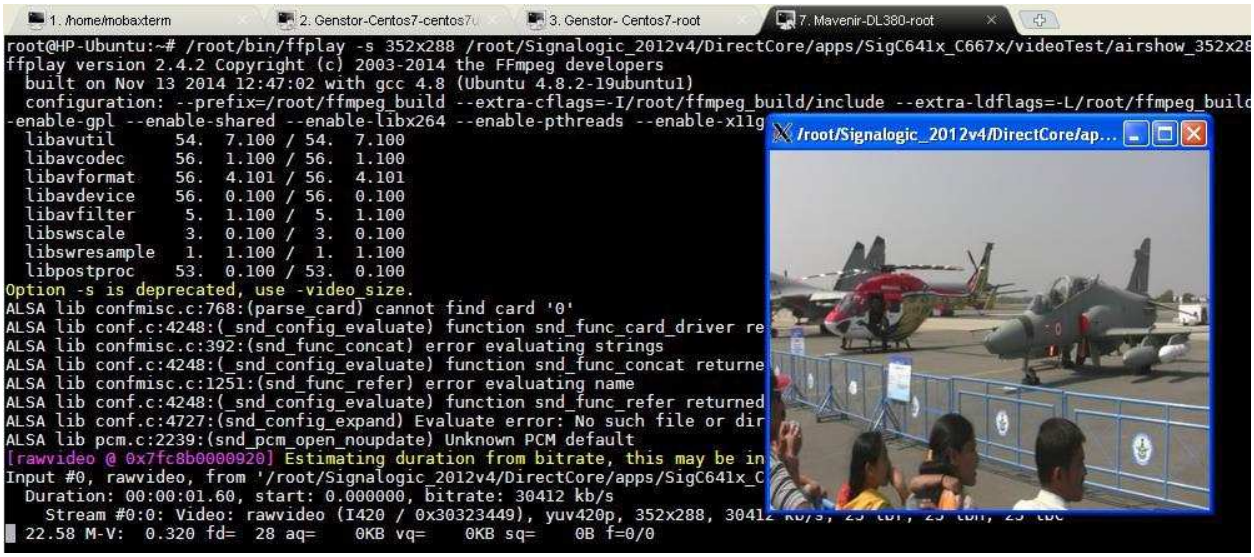

1. /home/mobaxterm
2. Genstor-Centos7-centos7
3. Genstor- Centos7-root
7. Mavenir-DL380-root
root@HP-Ubuntu:~# /root/bin/ffplay -s 352x288 /root/Signalogic_2012v4/DirectCore/apps/SigC641x_C667x/videoTest/airshow_352x28
ffplay version 2.4.2 Copyright (c) 2003-2014 the FFmpeg developers
built on Nov 13 2014 12:47:02 with gcc 4.8 (Ubuntu 4.8.2-19ubuntu1)
configuration: --prefix=/root/ffmpeg_build --extra-cflags=-I/root/ffmpeg_build/include --extra-ldflags=-L/root/ffmpeg_build
-enable-gpl --enable-shared --enable-libx264 --enable-pthreads --enable-x11g
libavutil 54. 7.100 / 54. 7.100
libavcodec 56. 1.100 / 56. 1.100
libavformat 56. 4.101 / 56. 4.101
libavdevice 56. 0.100 / 56. 0.100
libavfilter 5. 1.100 / 5. 1.100
libswscale 3. 0.100 / 3. 0.100
libswresample 1. 1.100 / 1. 1.100
libpostproc 53. 0.100 / 53. 0.100
Option -s is deprecated, use -video_size.
ALSA lib confmisc.c:768:(parse_card) cannot find card '0'
ALSA lib conf.c:4248:(_snd_config_evaluate) function snd_func_card_driver re
ALSA lib confmisc.c:392:(snd_func_concat) error evaluating strings
ALSA lib conf.c:4248:(_snd_config_evaluate) function snd_func_concat returne
ALSA lib confmisc.c:1251:(snd_func_refer) error evaluating name
ALSA lib conf.c:4248:(_snd_config_evaluate) function snd_func_refer returned
ALSA lib conf.c:4727:(snd_config_expand) Evaluate error: No such file or dir
ALSA lib pcm.c:2239:(snd_pcm_open_noupdate) Unknown PCM default
[rawvideo @ 0x7fc8b0000920] Estimating duration from bitrate, this may be in
Input #0, rawvideo, from '/root/Signalogic_2012v4/DirectCore/apps/SigC641x_C
Duration: 00:00:01.60, start: 0.000000, bitrate: 30412 kb/s
Stream #0:0: Video: rawvideo (I420 / 0x30323449), yuv420p, 352x288,
22.58 M-V: 0.320 fd= 28 aq= 0KB vq= 0KB sq= 0B f=0/0
/root/Signalogic_2012v4/DirectCore/ap...

Figure 4-15, Example of original (unencoded) video input .yuv file data, viewed with ffplay

The following .yuv files should be installed under the folder

```
install_path/Signalogic/video_files
```

and can be used with the streamTest demo to provide some examples of compression quality at different resolutions.

Filename	Resolution	Frame Rate	YUV Pixel Format	Comments
airshow_352x288p_30fps_420fmt.yuv	352 x 288	30	4:2:0 [1]	Very short (less than 5 sec) video with intermediate encoding challenge. The camera moves so video images are changing frame-to-frame, but the background is of medium complexity
touchdown_pass_1080p_30fps_420fmt.yuv	1920 x 1080	30	4:2:0	About 19 sec video clip with sports action (football game)
touchdown_pass_640x480p_30fps_420fmt.yuv	640 x 480	30	4:2:0	Same, but 640x480 resolution
parkrun_720p_50fps_420fmt.yuv	1280 x 720	50	4:2:0	About 11 sec video clip of a man running in a park. This video provides a difficult encoding challenge as (i) the background is complex (forest and park scenery) and (ii) the camera moves so the video images are changing frame-to-frame
hallway_352x288p_30fps_420fmt.yuv	352 x 288	30	4:2:0	Two people in a hallway carrying and picking up objects. This video is used for image analytics moving object tracking demos
akiyo_176x144p_30fps_420fmt.yuv	176 x 144	30	4:2:0	News anchor reading the news. This video may be used for some debug and test purposes

[1] 1.5 bytes/pixel

Table 4-1, streamTest example test videos

4.4.5 appTest

Note 1: ***This section assumes you have already (i) built and loaded the DirectCore target multicore CPU driver, as explained in section 4.1 above, Building the Loadable Kernel Module (LKM) Driver, and(ii) built all DirectCore libraries, as explained in section 4.2 above, Building DirectCore Libraries.***

Note 2: ***Currently appTest is only supported by SigC641x hardware. Building and/or running appTest should not be attempted for SigC667x hardware.***

The appTest program gives an example of communicating with target code as it runs in real-time, demonstrating several DirectCore library API calls.

The appTest program is written in C++ and uses Call Manager APIs to perform target-related functions and to reduce the number of API calls needed to initialize and control the target multicore CPU farm.

To build the appTest program, enter the following:

```
#cd /install_path/Signalogic/DirectCore/apps/SigC641x_C667x/apps/appTest

#make
```

Note that the libraries mentioned in section 4.2 above, including Session Manager, must be built before appTest can be built successfully.

To run the appTest program, first enter:

```
#./appTest –h
```

This will bring up a help menu. Command line option syntax (options with "+" sign are mandatory):

```
-c<string>      +    Card designator (e.g. -cSIGC64XX)

-f<num>              Device clock rate in MHz (e.g. -f1000)

-m<num>         +    Core select bit mask. (e.g. -m1, means
                     core0,-m2 means core1, -m255 means cores 0-
                     7, etc)

-e<string>      +    Target executable file (in ELF or COFF
                     format, e.g. -esigc64xx.out)
```

-p<num>	Number of devices (e.g. -p8)
-t	Run with TDM interface enabled
-C	Run with continuous diagnostic display enabled
-T	Run with target HPI driver ('talker') enabled
-v	Run in verbose mode
-h	Display this help list

For a 1 GHz SigC64x card, the default configuration used to run appTest would be:

```
#./appTest -m1 –f1000 -etmsc64xx.out -cSIGC64XX -t
```

For a 1.25 GHz SigC66x card, the default configuration used to run appTest would be:

```
#./appTest -m1 –f1250 -etmsc66xx.out -cSIGC66XX
```

Note that SigC66xx cards do not support a TDM interface. Note also that in the above command-line, the "-etmsc64xx.out" or "-etmsc66xx.out" option specifies a executable code file generated by Texas Instruments CGT (Code Generation Tools), v7.2.x or higher. See "Host Test Programs" section for a table showing the corresponding CCS projects and target executable code files for each host test program.

4.4.5.1 appTest Output Description

Figure 4-15 below is a screen capture of a successful appTest run, executed using the following command:

```
#./appTest -m1 -f720 -etmsc64xx.out -cSIGC64XX –t -C
```

The screen capture in Figure 4-15 below shows continuous target multicore CPU memory diagnostic data capture and display (enabled with –c command-line option) and results of the "display call list" command, showing several calls currently active.

 Copyright © Signalogic 2012-2015

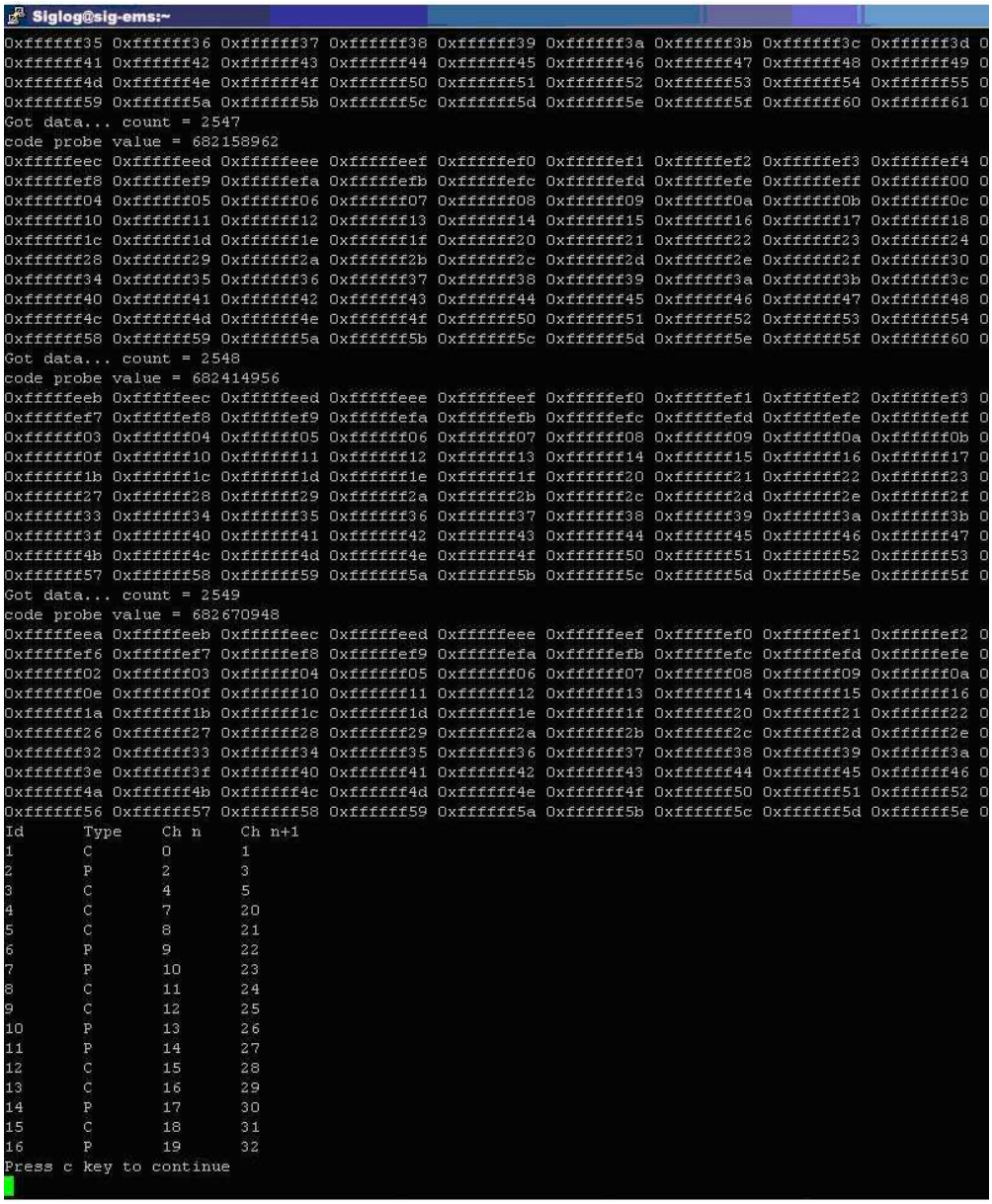

Siglog@sig-ems:~
0xffffff35 0xffffff36 0xffffff37 0xffffff38 0xffffff39 0xffffff3a 0xffffff3b 0xffffff3c 0xffffff3d 0
0xffffff41 0xffffff42 0xffffff43 0xffffff44 0xffffff45 0xffffff46 0xffffff47 0xffffff48 0xffffff49 0
0xffffff4d 0xffffff4e 0xffffff4f 0xffffff50 0xffffff51 0xffffff52 0xffffff53 0xffffff54 0xffffff55 0
0xffffff59 0xffffff5a 0xffffff5b 0xffffff5c 0xffffff5d 0xffffff5e 0xffffff5f 0xffffff60 0xffffff61 0
Got data... count = 2547
code probe value = 682158962
0xfffffeec 0xfffffeed 0xfffffeee 0xfffffeef 0xfffffef0 0xfffffef1 0xfffffef2 0xfffffef3 0xfffffef4 0
0xfffffef8 0xfffffef9 0xfffffefa 0xfffffefb 0xfffffefc 0xfffffefd 0xfffffefe 0xfffffeff 0xffffff00 0
0xffffff04 0xffffff05 0xffffff06 0xffffff07 0xffffff08 0xffffff09 0xffffff0a 0xffffff0b 0xffffff0c 0
0xffffff10 0xffffff11 0xffffff12 0xffffff13 0xffffff14 0xffffff15 0xffffff16 0xffffff17 0xffffff18 0
0xffffff1c 0xffffff1d 0xffffff1e 0xffffff1f 0xffffff20 0xffffff21 0xffffff22 0xffffff23 0xffffff24 0
0xffffff28 0xffffff29 0xffffff2a 0xffffff2b 0xffffff2c 0xffffff2d 0xffffff2e 0xffffff2f 0xffffff30 0
0xffffff34 0xffffff35 0xffffff36 0xffffff37 0xffffff38 0xffffff39 0xffffff3a 0xffffff3b 0xffffff3c 0
0xffffff40 0xffffff41 0xffffff42 0xffffff43 0xffffff44 0xffffff45 0xffffff46 0xffffff47 0xffffff48 0
0xffffff4c 0xffffff4d 0xffffff4e 0xffffff4f 0xffffff50 0xffffff51 0xffffff52 0xffffff53 0xffffff54 0
0xffffff58 0xffffff59 0xffffff5a 0xffffff5b 0xffffff5c 0xffffff5d 0xffffff5e 0xffffff5f 0xffffff60 0
Got data... count = 2548
code probe value = 682414956
0xffffffeeb 0xffffffeec 0xffffffeed 0xffffffeee 0xffffffeef 0xffffffef0 0xffffffef1 0xffffffef2 0xffffffef3 0
0xffffffef7 0xffffffef8 0xffffffef9 0xffffffefa 0xffffffefb 0xffffffefc 0xffffffefd 0xffffffefe 0xffffffeff 0
0xffffff03 0xffffff04 0xffffff05 0xffffff06 0xffffff07 0xffffff08 0xffffff09 0xffffff0a 0xffffff0b 0
0xffffff0f 0xffffff10 0xffffff11 0xffffff12 0xffffff13 0xffffff14 0xffffff15 0xffffff16 0xffffff17 0
0xffffff1b 0xffffff1c 0xffffff1d 0xffffff1e 0xffffff1f 0xffffff20 0xffffff21 0xffffff22 0xffffff23 0
0xffffff27 0xffffff28 0xffffff29 0xffffff2a 0xffffff2b 0xffffff2c 0xffffff2d 0xffffff2e 0xffffff2f 0
0xffffff33 0xffffff34 0xffffff35 0xffffff36 0xffffff37 0xffffff38 0xffffff39 0xffffff3a 0xffffff3b 0
0xffffff3f 0xffffff40 0xffffff41 0xffffff42 0xffffff43 0xffffff44 0xffffff45 0xffffff46 0xffffff47 0
0xffffff4b 0xffffff4c 0xffffff4d 0xffffff4e 0xffffff4f 0xffffff50 0xffffff51 0xffffff52 0xffffff53 0
0xffffff57 0xffffff58 0xffffff59 0xffffff5a 0xffffff5b 0xffffff5c 0xffffff5d 0xffffff5e 0xffffff5f 0
Got data... count = 2549
code probe value = 682670948
0xffffffeea 0xffffffeeb 0xffffffeec 0xffffffeed 0xffffffeee 0xffffffeef 0xffffffef0 0xffffffef1 0xffffffef2 0
0xffffffef6 0xffffffef7 0xffffffef8 0xffffffef9 0xffffffefa 0xffffffefb 0xffffffefc 0xffffffefd 0xffffffefe 0
0xffffff02 0xffffff03 0xffffff04 0xffffff05 0xffffff06 0xffffff07 0xffffff08 0xffffff09 0xffffff0a 0
0xffffff0e 0xffffff0f 0xffffff10 0xffffff11 0xffffff12 0xffffff13 0xffffff14 0xffffff15 0xffffff16 0
0xffffff1a 0xffffff1b 0xffffff1c 0xffffff1d 0xffffff1e 0xffffff1f 0xffffff20 0xffffff21 0xffffff22 0
0xffffff26 0xffffff27 0xffffff28 0xffffff29 0xffffff2a 0xffffff2b 0xffffff2c 0xffffff2d 0xffffff2e 0
0xffffff32 0xffffff33 0xffffff34 0xffffff35 0xffffff36 0xffffff37 0xffffff38 0xffffff39 0xffffff3a 0
0xffffff3e 0xffffff3f 0xffffff40 0xffffff41 0xffffff42 0xffffff43 0xffffff44 0xffffff45 0xffffff46 0
0xffffff4a 0xffffff4b 0xffffff4c 0xffffff4d 0xffffff4e 0xffffff4f 0xffffff50 0xffffff51 0xffffff52 0
0xffffff56 0xffffff57 0xffffff58 0xffffff59 0xffffff5a 0xffffff5b 0xffffff5c 0xffffff5d 0xffffff5e 0
Id Type Ch n Ch n+1
1 C 0 1
2 P 2 3
3 C 4 5
4 C 7 20
5 C 8 21
6 P 9 22
7 P 10 23
8 C 11 24
9 C 12 25
10 P 13 26
11 P 14 27
12 C 15 28
13 C 16 29
14 P 17 30
15 C 18 31
16 P 19 32
Press c key to continue

Figure 4-15, Example appTest display

appTest Interactive Keyboard Commands (showing Main Level commands only):

A	Algorithm Mode Control
C	Show list of currently active call IDs
D	Display diagnostic data (currently shows per-channel state and flag information, target code probe points, and IP/UDP/RTP packet information)

F	Freeze/unfreeze continuous target data display
L	Load media
M	Show list of currently active media IDs
P	Play media
O	Operating Mode Control
S	Set up call
T	Tear down call
U	Unload media
Q	Quit

***NOTE:** The above command summary is only an abbreviated list given for example and reference purposes. This summary should be solely for quick test runs of the appTest program. For detailed information on appTest program interactive commands, see the following section 4.3.5, Voice and Call Processing Test Procedures.*

4.4.5.2 RTP Audio Test Program

The RTP Audio test program is a WinXP program that:

- Receives UDP/RTP streams from the network and plays to the sound card (headphones or speaker), and
- Transmits sound card input data (microphone) or waveform file data (.wav file) over the network via UDP/RTP streams

The RTP Audio program can be set to work in different modes, including half duplex, full duplex, .wav file data or microphone source, and with flexible RTP parameters, such as 16-bit linear and 8-bit compressed payload formats and variable payload lengths.

The following are some command line examples:

1. Here the command line specifies source data as microphone and destination as a UDP/RTP stream to IP address 10.0.0.53, using 8-bit uLaw RTP payloads:

```
-smic –d10.0.0.53 -u8
```

2. Here the command line specifies source data as a UDP/RTP stream received from a remote PC with IP address 10.0.0.53, and destination as the sound card, using 16-bit linear PCM format (also known as “L16”):

```
-s10.0.0.53 -dsnd
```

3. The following command lines set up an “RTP call” between two PC’s, allowing full-duplex, real-time audio communication via UDP/RTP streams:

```
-smic -d10.0.0.53 -u8 -f

-s10.0.0.29 -dsnd -u8 -f
```

4.4.5.2.1 Installing and Running RTP Audio

1. To install the RTP Audio test program, download “WinXP_01Apr09.rar” or “RtpAudio.zip” or similar compressed file from Signalogic’s website, and extract to a suitable folder on your target PC. For full-duplex operation between two PCs, two copies of RtpAudio should be installed, on one each PC.

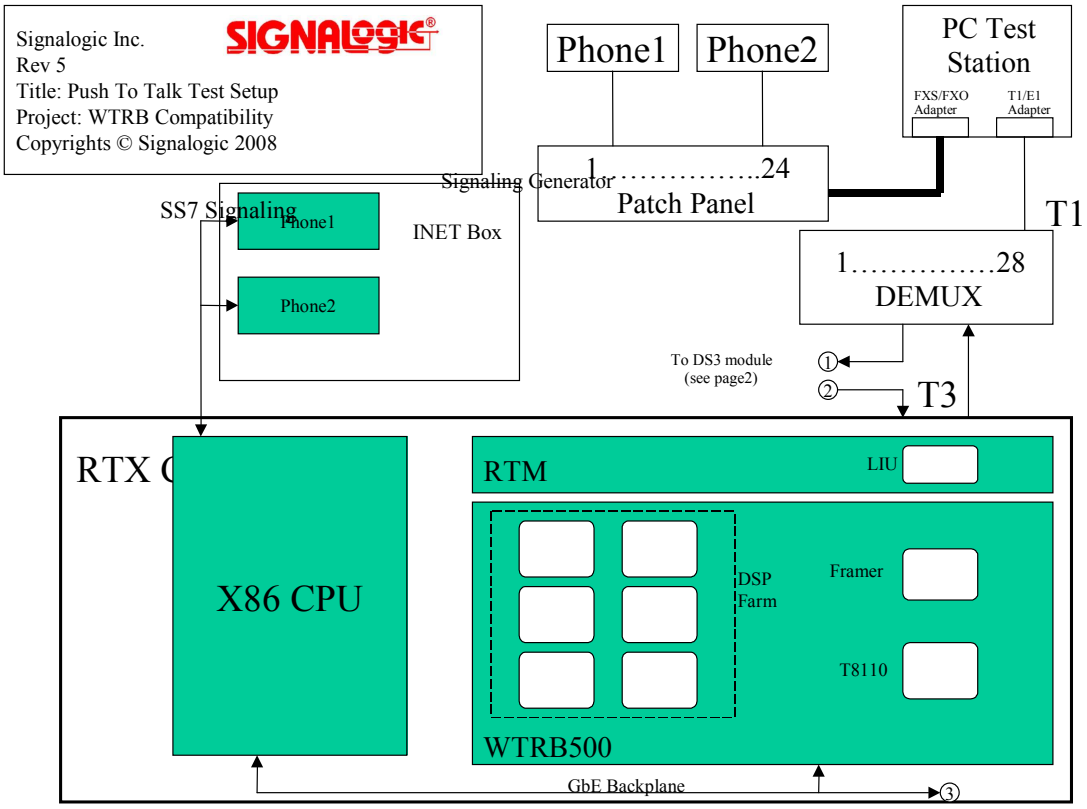
2. To run the RTP Audio test program, open a console application window (e.g. run “Cmd” under WinXP), go to the directory on which you installed the executable, and run “rtpaudio” followed by command-line entries such as those shown above.

Note: For detailed information about the RTP Audio Test Program, please refer to the “RTP Audio Test Program Users Guide”, which contains system diagrams, program usage screen captures, RTP payload and header tables, and more.

4.4.5.3 Call and Voice Processing Test Procedures

***Note:* This section only applies if a TDM interface is being used. If not, ignore this section and skip to section 4.3.6, Interactive Command Summary.**

The test procedures given in this section assume a test setup as shown in Figure 4-16 below:



Page 1/2 Note: The HW400c2 and WTRB500 are installed in the same RTX chas sis

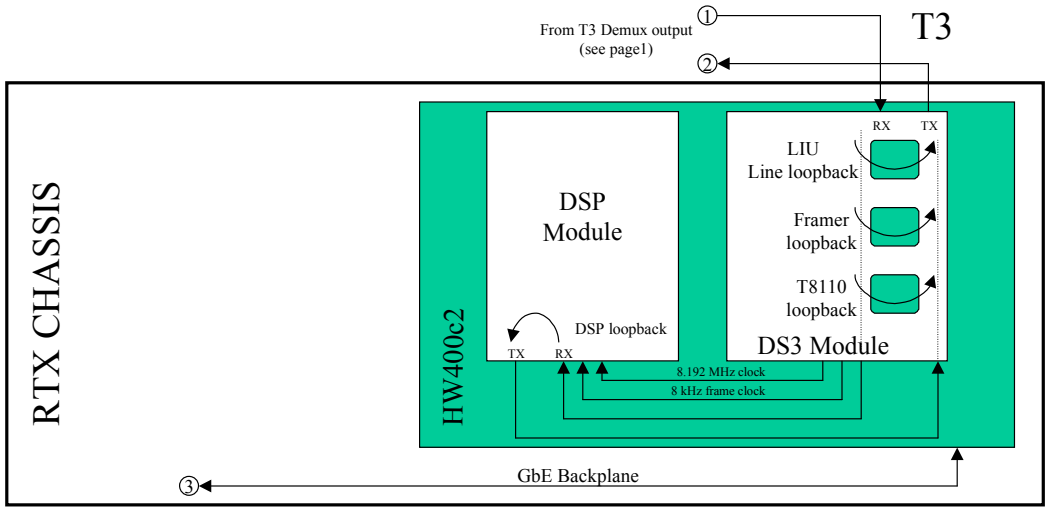


Figure 4-16, TDM call equipment and setup

In Figure 4-16 above, “Phone 1” and “Phone 2” refer to standard handsets with RJ-11 connectors. The handsets are used for verbal and aural confirmations in the test procedures below.

4.4.5.4 TDM Setup Procedures

1. First make sure both SigC641x module and DS3 module drivers (module names sigc641x and wan256t3, respectively) are loaded.

Test program sequences below will fail immediately if both drivers are not loaded and functioning correctly. For the SigC641x module driver, the command lines to do this are:

```
cd ../BaseFolder/SigC641x/driver

make load
```

To make (build) the DS3 module driver (wan256t3), type the following:

```
cd /usr/src/linux/

make modules

make modules_install
```

To load the DS3 driver, type:

```
modprobe wan256t3
```

 Copyright © Signalogic 2012-2015

To unload the driver, type:

```
rmmod wan256t3
```

To see driver options, type:

```
modinfo wan256t3
```

2. Run the DS3cfg script, using the following command line sequence:

```
cd ../BaseFolder/DS3

./DS3cfg
```

The DS3cfg script performs basic configuration and initialization of the TECT3 framer and T8110 devices on the DS3 module. With the T8110, all channel mapping and connections are reset (cleared). If a digital scope is connected to the CTBus between the DS3 module and SigC641x module, as shown in Figure 4-17 below, then the CTD0 and CTD1 traces should show "flat lines" (no data). Even though T3 input/output to the DS3 module may be active, and TECT3 framer on the DS3 module configured as needed, without T8110 channel connections the no data will be present on the CTBus.

Note: TECT3 framer and T8110 chip setup contained in the DS3cfg script is progressively being replaced by test program code. Running the script beforehand is always a safe step, even if the test program Resets these chips and rewrites chip registers.

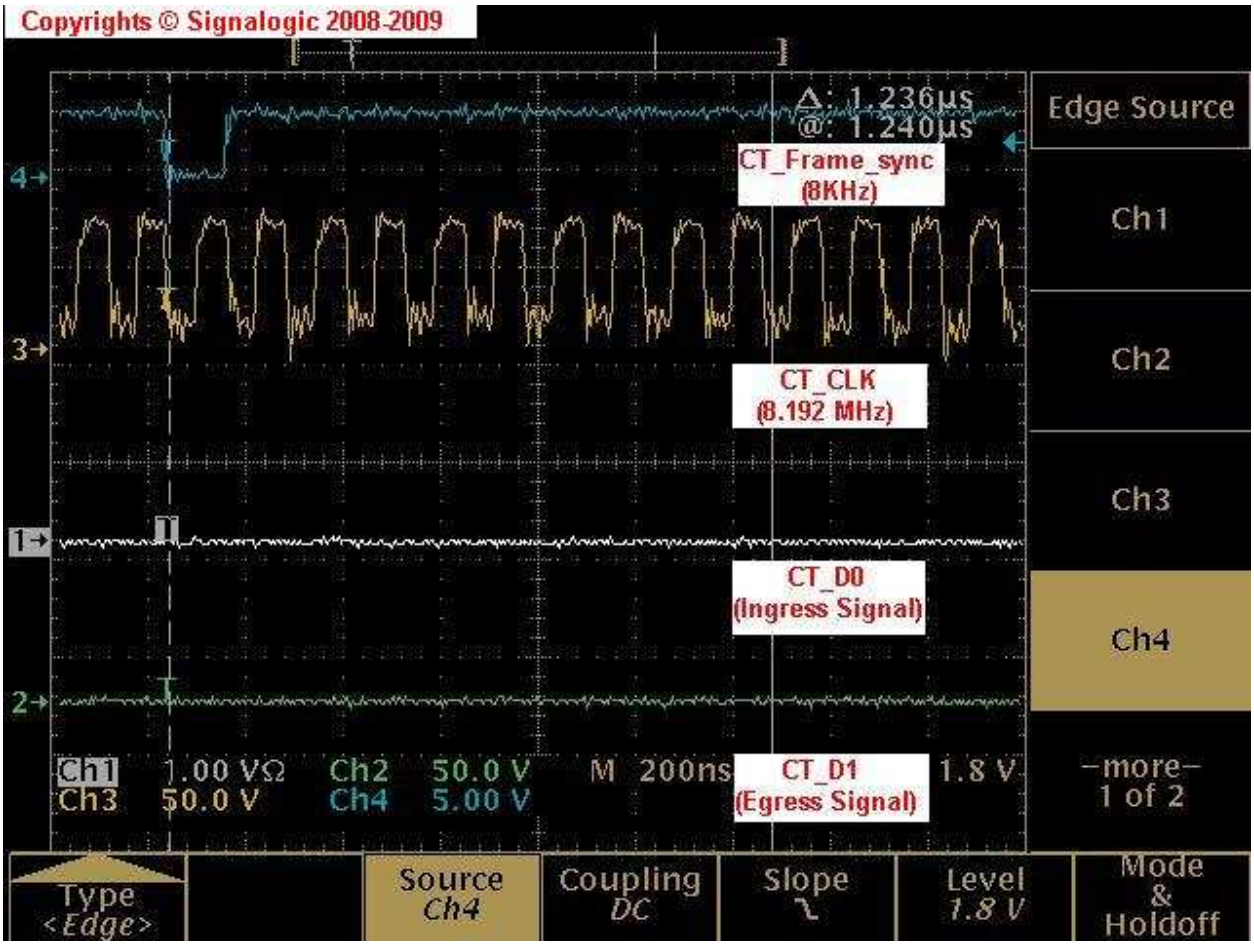


Figure 4-17, CTBus between SigC641x and DS3 modules, with voice data inactive

In the appTest program, no connections are made until a call is set up. If for testing or debug purposes connections are needed prior to call setup, the DS3cfg script can be modified. A version of the script called DS3cfg_mapping is available as an example of how to do this – look near the end of the script.

3. ****Note – this step is optional. Updated test procedures do not require this step.**** Initialize a call using the CPU blade software, WTRB500 media board, and Inet signaling test/generation unit (connected to SS7 signaling module located on the CPU blade).

At least one call has to be initialized in the system to ensure valid T3 output from the WTRB500 board. After this is done, the T3 cabling should be adjusted so the WTRB500 output is bypassed; in other words, T3 output from the Adtran MX2800 demux output should be routed to both the WTRB500 board and DS3 module (use a splitter connector) and DS3 module T3 output should be routed back to the MX2800 mux input.

4. Run the appTest program, using the following command line:

```
cd /install_path/Signalogic/DirectCore/apps/SigC641x_C667x/appTest

#./appTest -m1 -f720 -etmsc64xx.out -cSIGC64XX –t
```

The current version of the test program will initially show a substantial amount of debug information related to hardware and driver initialization, and pause three (3) times for keyboard prompt. At the first two prompts, push a space key, and at the third prompt enter a ‘c’ key (“Continue”) or ‘q’ key to Quit if debug information shows any type of hardware initialization error.

The test program will initialize with:

- Target voice code loopback on all channels (logic loopback disabled)
- TDM mapping set to “non reversed”, which assumes the Motorola WTRB500 board is not present in the T3 circuit
- No calls connected
- No medias downloaded into target multicore CPU memory

4.4.5.5 Interactive Command Summary

Below is a summary of interactive key commands available when running the test program. Commands are in two levels, a Main level and a Second level as shown:

Main Level	Second Level	Description
A		Algorithm Mode Control
	A	Exit algorithm mode control
	D	Display VAD and DTMF status
	E	Toggle Echo Reduce (default = active)
	G	Toggle AGC (default = active)
	T	Toggle DTMF (default = active)
	V	Toggle VAD (default = active)

 Copyright © Signalogic 2012-2015

C		Show list of currently all active IDs
D		Display diagnostic data (currently shows per-channel state and flag information, taret probe points, and IP/UDP/RTP packet information)
F		Freeze/unfreeze continuous target data display
L		Load media
M		Show list of currently active media IDs
O		Operating Mode Control
	O	Exit operating mode control
	D	Enable target voice processing (default = enabled)
	I	Prompt for entry of IP address for IP call testing
	L	Enable logic loopback of CTBus streams (default = disabled)
	N	Enable CTBus stream registers (default = enabled)
	P	Toggle debug prompt enable
	>	Toggle IP Test Mode (default = enabled)
	W	Toggle whether WTRB500 board is in series in the T3 circuit (default is not in the circuit)
	V	Toggle verbose mode for diagnostic data display (default is disabled)
P		Play media
S		Set up call
T		Tear down call

 Copyright © Signalogic 2012-2015

U		**Unload media**
Q		**Quit**

4.4.5.6 Basic Call, Media, and DTMF Test Procedures

***Note:** This section includes references to a TDM interface (e.g. CTBus). If no TDM interface is installed or configured, those references may be ignored. References to IP calls, DTMF events, and Media playouts, and interactive command interface remain valid.*

1. First enter Operating Mode (press 'o' key) and verify that if you alternately press the 'n' and 'd' keys you can alternately not hear / hear voice loopback on any connected phone (by default, loopback is enabled on all channels on all streams).

2. If you have a digital scope connected as shown in Figure 4-18 below, you can press 'L' key (still within Operating Mode) to verify that with Logic Loopback enabled, input and output CTD0 and CTD1 traces show identical bit patterns. This only occurs if no delay is present in the CTBus circuit, which is the case with logic loopback, but not with target voice processing. Press the 'd' key to re-enable target voice processing.

3. If in Operating Mode, then exit by pressing the 'o' key again. Use the 's' key to set up calls. The call setup prompt entry is of the format (brackets indicate optional entry):

```
type[Id].a.b[.dsp]
```

where type is Call Type (either 'c' for conference or 'p' for PTT) Id is an optional Call Id, a and b specify call leg information, and dsp is a target CPU number (from zero to 7).

A and b can specify:

- Two call legs to create a 2-party call, with a and b each specifying a call leg in the form of a channel or endpoint. Some examples:

`c.0.1`	Creates a conference call having Call Id of 1, channels 0 and 1
`p.e3.e4`	Creates a PTT call having a Call Id of 2, with the talker channel initially assigned to endpoint 3, and first listener channel assigned to endpoint 4

- One call leg, when a and b both specify a channel and/or endpoint. In this case, the Call Id must be given. Some examples:

`c3.4.e5`	Create a call leg on channel 4, endpoint 5, with Call Id assigned a value of 3

`p4.5.e6`	Create a PTT talker call leg on channel 5, endpoint 6, with Call Id assigned a value of 4

Channel format can be:

```
N
```

or

```
iN
```

where N is a channel number from 0 to 127 and 'i' specifies an IP channel (default with no prefix specified is a TDM channel). For an IP channel, no endpoint or T1 stream information should be entered.

Endpoint format can be:

```
eNtM
```

where 'e' indicates an "endpoint" (time-slot on a T1 stream, from 1 to 24), 't' indicates call leg source is T1, and M is a T1 stream number from 1 to 28. For example, an explicit entry of:

```
c.e1t1.e1.t2
```

would set up a call between endpoint 1 on T1 stream #1 and endpoint 1 on T1 stream #2.

If a target CPU number is not specified, then the "currently active" DSP is assumed. Normally this is zero (first target CPU).

Note that partial entry is allowed for call and call leg input. For example, when creating calls, if no endpoint information is given, then the first available endpoint is assigned. If no channel is specified then the first available channel is assigned. If no call leg source information is given, then the first available T1 is assumed.

Call legs may be added to an existing call by specifying an existing Call Id. Given the above examples, the following commands would add a third leg to Call Ids 3 and 4:

`c3.9.e21`	Add a leg to conference Call Id 3, assigned to channel 9 and endpoint 21 (on T1 stream #1).
`p4.27.e1t2`	Add a listener leg to PTT Call Id 4, assigned to channel 27 and endpoint 1 on T1 stream #2

With the above information as a guide, use the 's' key to set up the following six (6) calls:

```
c.0.1
```

 Copyright © Signalogic 2012-2015

```
c.e3.e4

c.e9.e10

c.6.7

c5.8.e8

c10.20.e20

c10.21.e21
```

The above sequence will set up six (6) calls. In the first call, endpoints are not specified so endpoints 1 and 2 are allocated. In calls 2 and 3, channels are not specified, so channels 2-5 are allocated. In the fourth (4^{th}) call, endpoints 5 and 6 are assigned, as they are the "next available" endpoints. In the fifth line above, a new call leg is created with channel 8 and endpoint 8 specified, and Call Id 5 assigned (given explicitly). In the sixth call, each leg is set up separately. Notes about the above call setups:

Call Ids 1 thru 4 are assigned for first 4 calls, a Call Id value of 5 is given explicitly for the fifth call, but only one leg of the call is active, so this call ID is reserved. A Call Id value of 10 is given explicitly for the sixth call

Only one call can be set up each time 's' is pressed

Call Id values are "one based". The maximum Call Id value is 255

If a digital scope is connected to the CTBus between the DS3 module and SigC641x modules, then the scope should show a display similar to Figure 4-18 below.

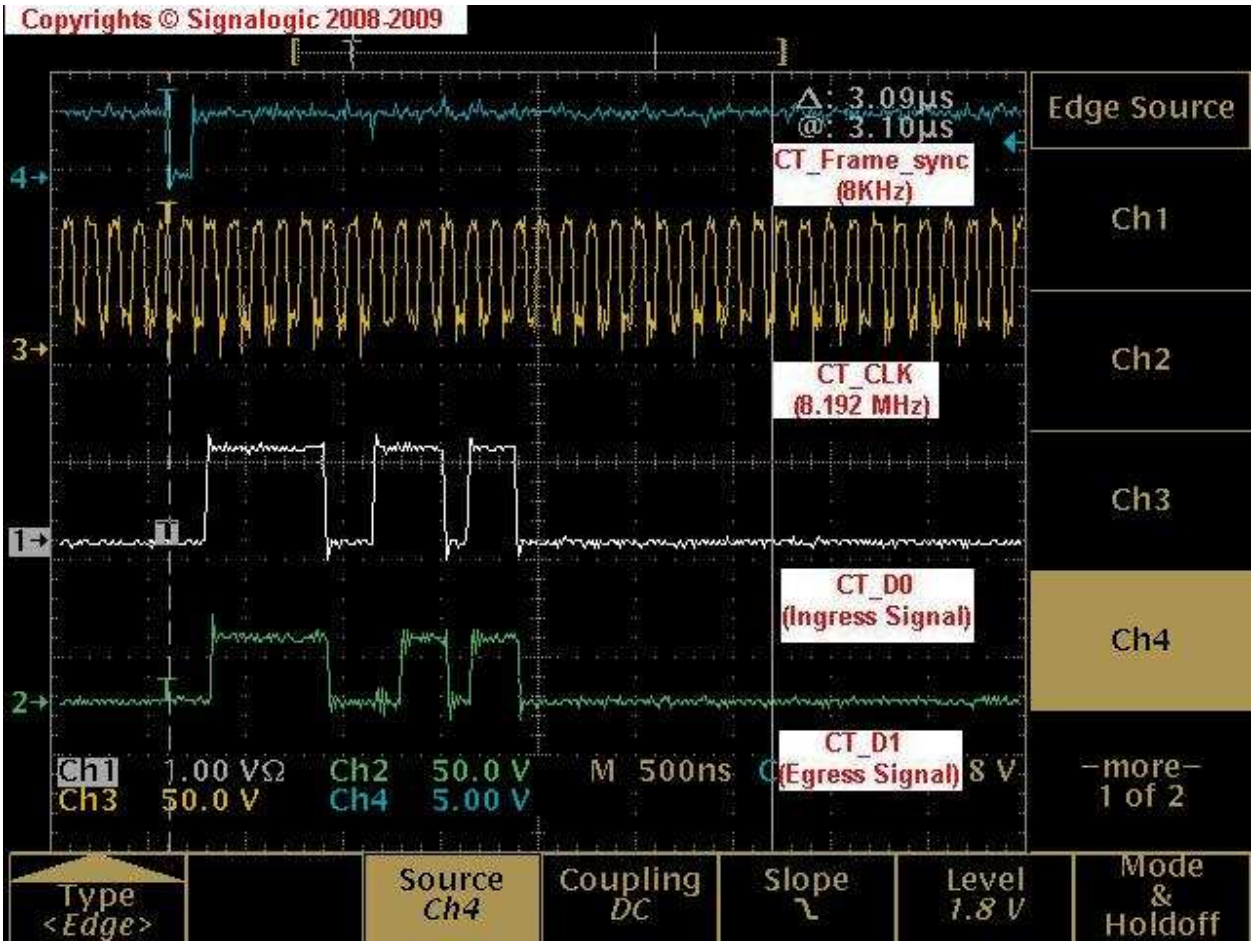


Figure 4-18, CTBus between SigC641x and DS3 modules, with voice data active

4. A call type of 'p' can be used to set up a PTT call. The following entry:

```
p.10.11
```

will set up a PTT call on channels 10 and 11 (using available endpoints 7 and 11), with the PTT "talker" assigned to channel 10 and the first PTT "listener" assigned to channel 11. When a PTT call is first established, a "PTT Buffering Start" event should be displayed. There are two types of this event: a) VAD, and b) Delay. The VAD version indicates that PTT talker voice was detected and PTT buffering started upon voice detection. The Delay version indicates that PTT buffering started after some non-voice when no PTT talker voice was detected (currently this includes a "dead channel" which is useful for lab testing, and a user-specified arbitrary delay).

Upon creation of the PTT call, a "PTT buffering start (VAD)" event should be displayed. About 8 seconds after the buffering start event, a "PTT Buffering Wrap" event should be displayed. Figure 4-19 below shows examples of PTT Buffering Start and Done events:

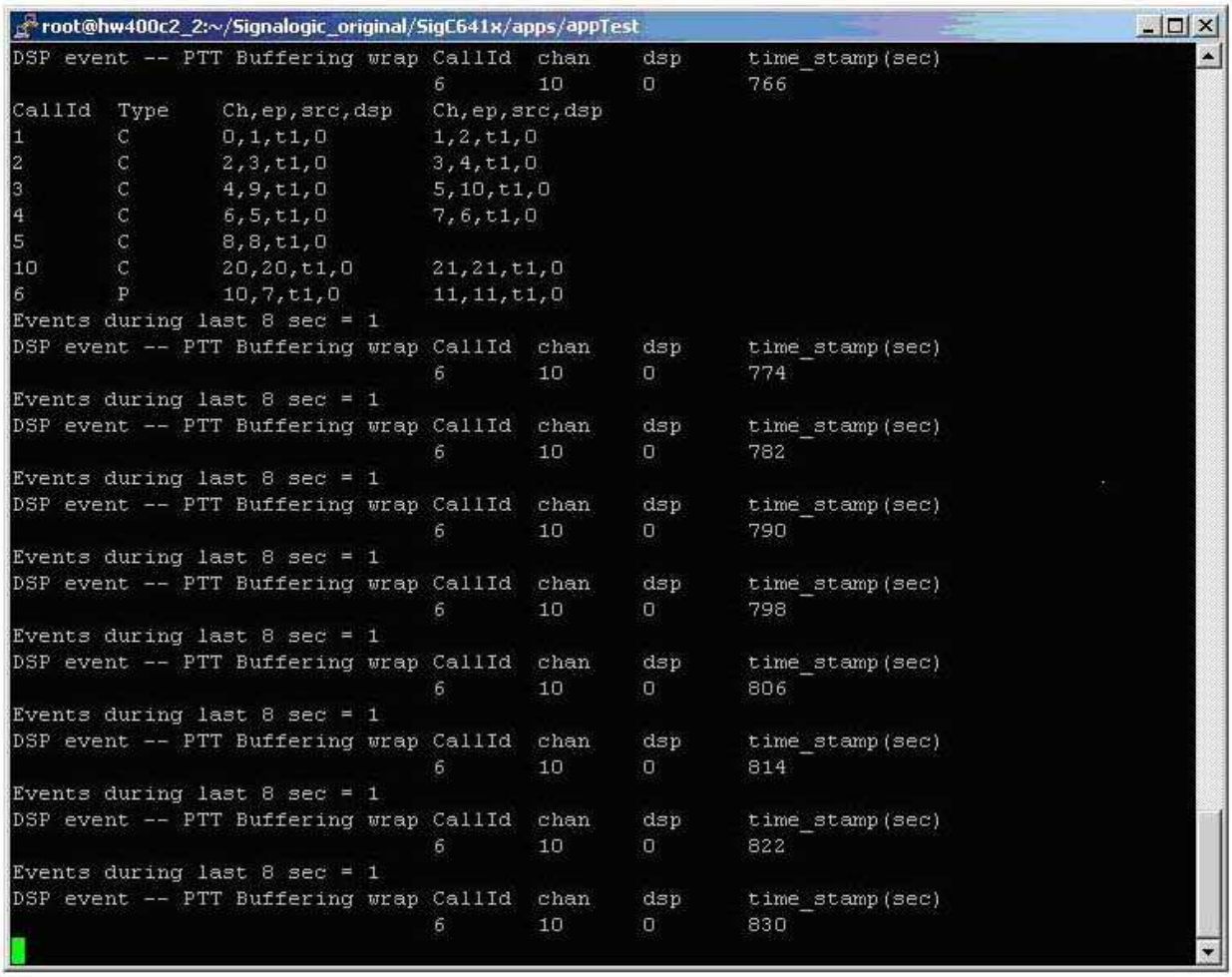

root@hw400c2_2:~/Signalogic_original/SigC641x/apps/appTest
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 766
CallId Type Ch,ep,src,dsp Ch,ep,src,dsp
1 C 0,1,t1,0 1,2,t1,0
2 C 2,3,t1,0 3,4,t1,0
3 C 4,9,t1,0 5,10,t1,0
4 C 6,5,t1,0 7,6,t1,0
5 C 8,8,t1,0
10 C 20,20,t1,0 21,21,t1,0
6 P 10,7,t1,0 11,11,t1,0
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 774
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 782
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 790
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 798
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 806
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 814
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 822
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 830

Figure 4-19, PTT Buffering Start and Done events after PTT call setup

5. Adding legs to an existing call. The following entry:

```
p6.12.e13
```

will add a third leg to PTT call with Call Id value of 6 (note that the value of 6 was assigned in the ‘p.10.11’ call setup entry in step 4 above, as 6 was the next available Call Id). This call leg will become the second PTT “listener” on the call. Multiple listeners can be assigned to a PTT call.

6. Now press 'c' key to verify the active call list. The test program should display currently active calls, as shown in Figure 4-20 below.

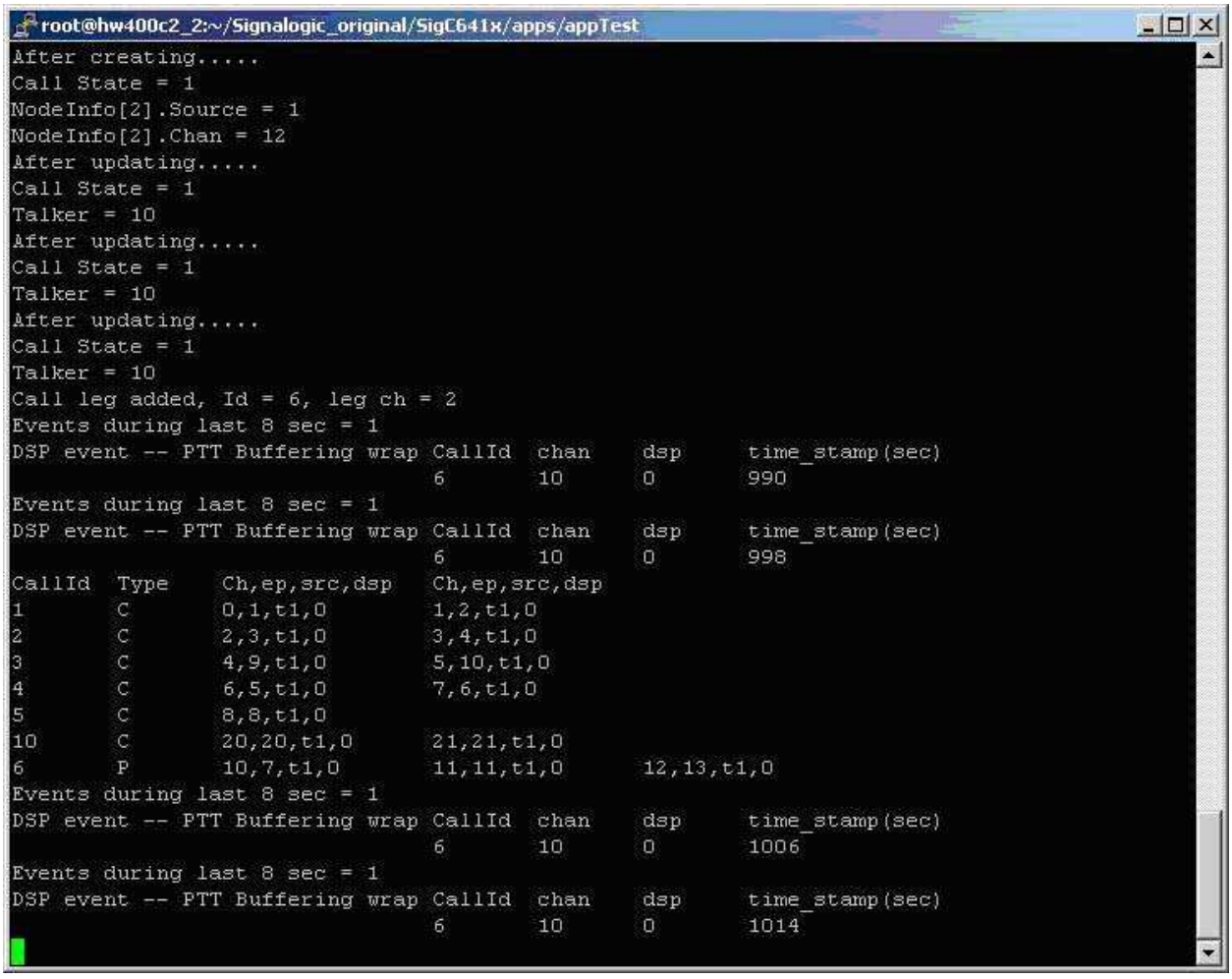

root@hw400c2_2:~/Signalogic_original/SigC641x/apps/appTest
After creating.....
Call State = 1
NodeInfo[2].Source = 1
NodeInfo[2].Chan = 12
After updating.....
Call State = 1
Talker = 10
After updating.....
Call State = 1
Talker = 10
After updating.....
Call State = 1
Talker = 10
Call leg added, Id = 6, leg ch = 2
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 990
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 998
CallId Type Ch,ep,src,dsp Ch,ep,src,dsp
1 C 0,1,t1,0 1,2,t1,0
2 C 2,3,t1,0 3,4,t1,0
3 C 4,9,t1,0 5,10,t1,0
4 C 6,5,t1,0 7,6,t1,0
5 C 8,8,t1,0
10 C 20,20,t1,0 21,21,t1,0
6 P 10,7,t1,0 11,11,t1,0 12,13,t1,0
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1006
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1014

Figure 4-20, Active call list display

Note that Call Id's are assigned in the order of call setup, unless Call Id is specified explicitly during the call setup entry. When a call is deleted (using the tear down command), then the deleted Call Id is returned to zero and made available again. Other active Call Id's are not changed when a call is deleted.

7. Updating a call. The following entry:

```
p.14.15
```

would first establish a PTT call on channels 14 and 15 (using next available endpoints of 15 and 16), with channel 14 (endpoint 15) acting as the talker leg and channel 15 (endpoint 16) the listener leg. The following entry:

```
c7.14.e15
```

will convert the PTT call to a 2-way conference call. Note that the channel + endpoint entry specification *exactly* matches the talker leg of the existing call.

8. Using test system equipment as shown in Figure 4-16 above, or similar equipment, verify one or more currently active TDM calls using handset phones.

9. To set up IP calls, take the following steps. First, enter source and remote IP addresses and UDP ports as shown below:

   a. press ‘o’ to see operating control mode options

   b. Choose ‘>’ to disable IP-to-IP test mode

   c. Type ‘i’ to enter a source IP address, press the ‘Enter’ key, and enter a remote IP address

   d. Type ‘o’ to exit operating control mode

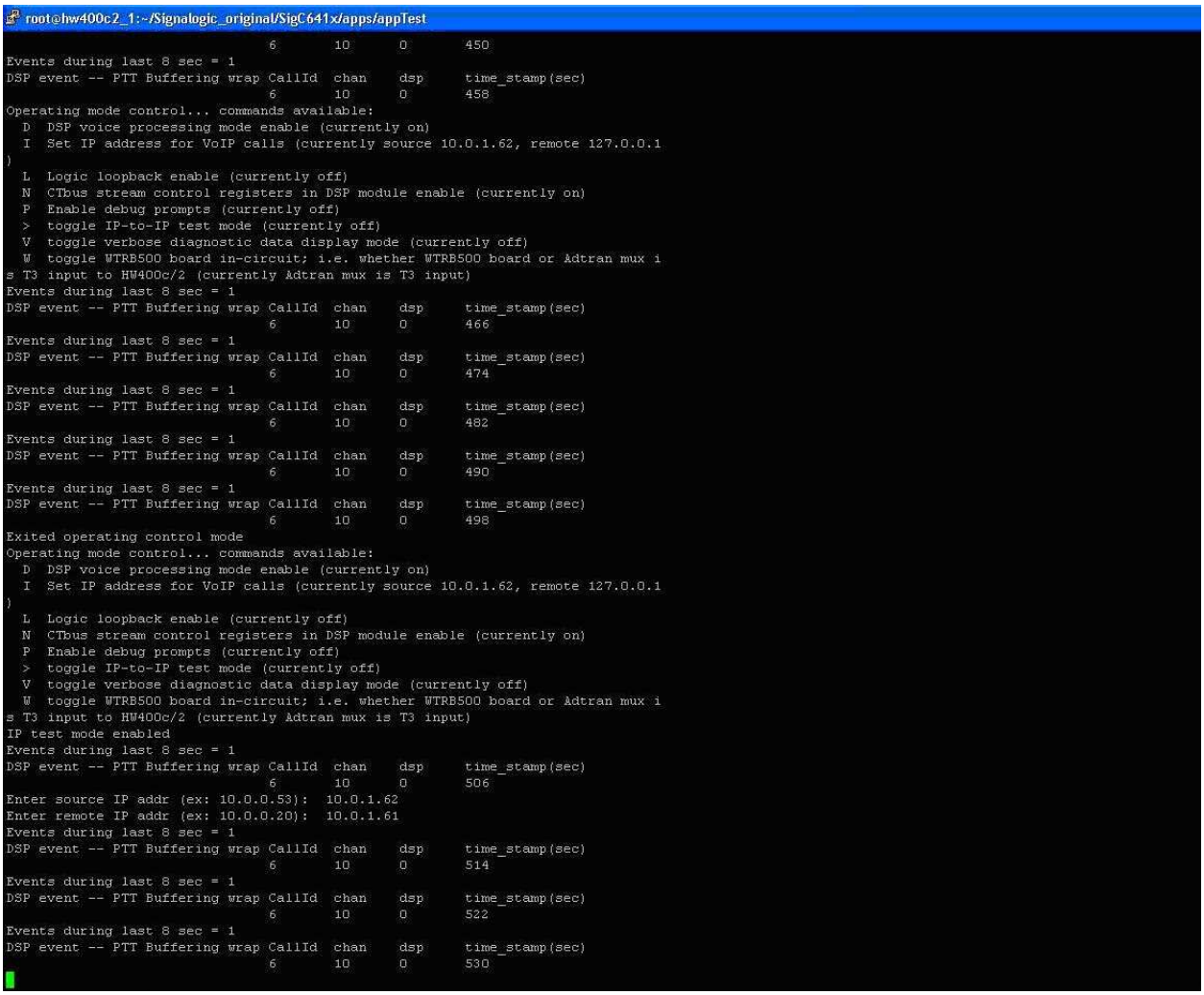

root@hw400c2_1:~/Signalogic_original/SigC641x/apps/appTest
6 10 0 450
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 458
Operating mode control... commands available:
D DSP voice processing mode enable (currently on)
I Set IP address for VoIP calls (currently source 10.0.1.62, remote 127.0.0.1
)
L Logic loopback enable (currently off)
N CTbus stream control registers in DSP module enable (currently on)
P Enable debug prompts (currently off)
> toggle IP-to-IP test mode (currently off)
V toggle verbose diagnostic data display mode (currently off)
W toggle WTRB500 board in-circuit; i.e. whether WTRB500 board or Adtran mux i
s T3 input to HW400c/2 (currently Adtran mux is T3 input)
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 466
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 474
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 482
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 490
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 498
Exited operating control mode
Operating mode control... commands available:
D DSP voice processing mode enable (currently on)
I Set IP address for VoIP calls (currently source 10.0.1.62, remote 127.0.0.1
)
L Logic loopback enable (currently off)
N CTbus stream control registers in DSP module enable (currently on)
P Enable debug prompts (currently off)
> toggle IP-to-IP test mode (currently off)
V toggle verbose diagnostic data display mode (currently off)
W toggle WTRB500 board in-circuit; i.e. whether WTRB500 board or Adtran mux i
s T3 input to HW400c/2 (currently Adtran mux is T3 input)
IP test mode enabled
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 506
Enter source IP addr (ex: 10.0.0.53): 10.0.1.62
Enter remote IP addr (ex: 10.0.0.20): 10.0.1.61
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 514
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 522
Events during last 8 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 530

Figure 4-21, Setting up an IP call

10. Next press ‘s’ to set up an IP call using the following command line:

```
c.i9.13
```

After setting up the call, press ‘Enter’, you should see the following screen.

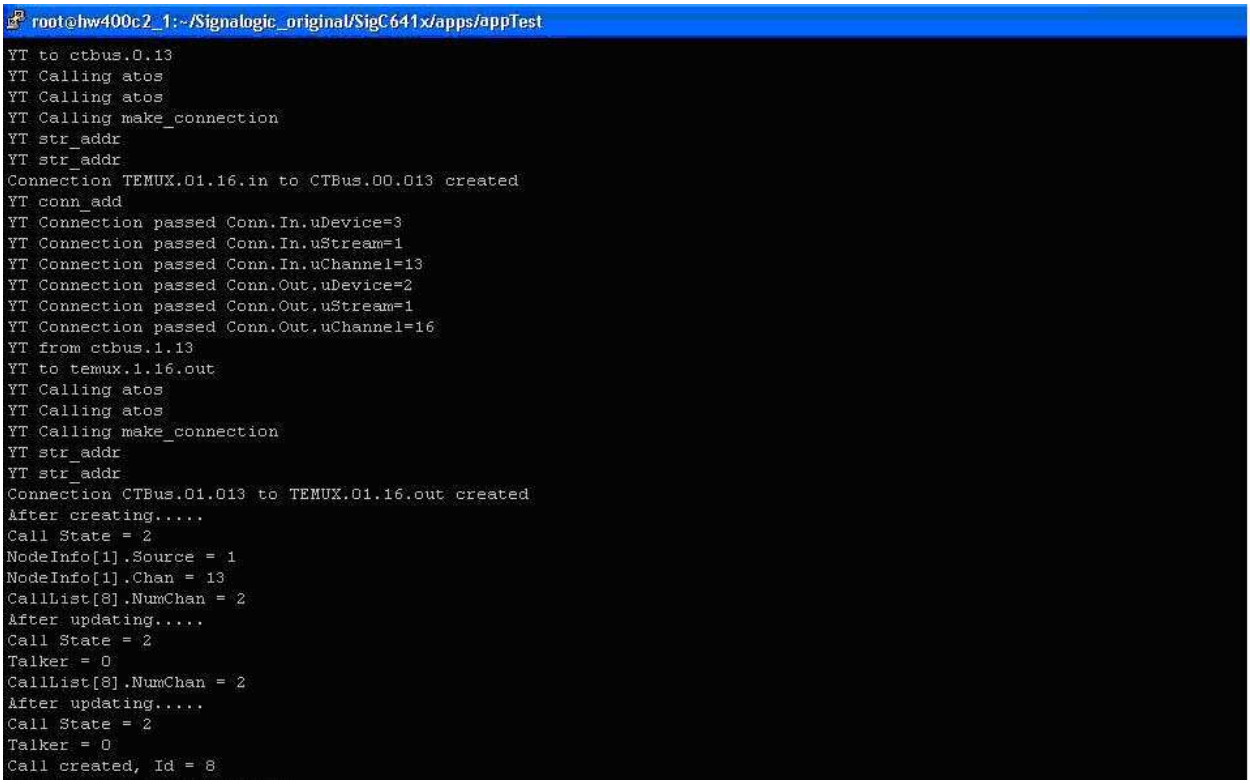


Figure 4-22, After setting up a TDM-to-IP call

11. Set up three (3) more IP calls as shown below:

```
c.16.i17

c.18.i19

c.i22.23
```

12. Now press 'c' key to verify the current call list. The test program should display all currently active calls, as shown in Figure 4-23 below. Note that IP channels are indicated with “ip” in the “src” column.

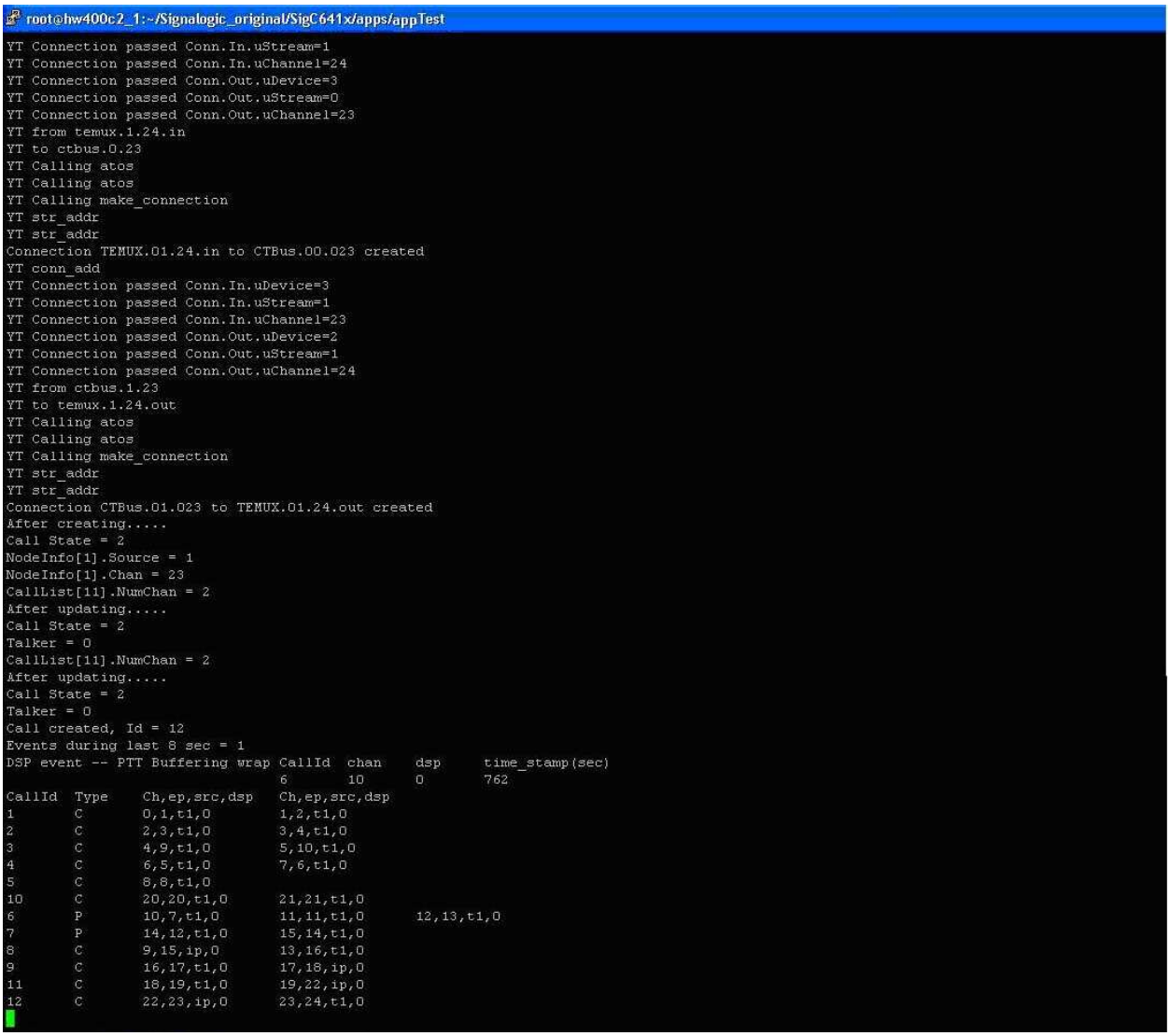


Figure 4-23, Call display including IP calls

13. Run “RTP Audio” test program on a separate WinXP machine to verify IP calls, using handsets (for TDM side of call) and headsets connected to the WinXP PC sound card (for IP side of call). For example run this RTP Audio command line:

```
RtpAudio -s10.0.1.62:16384 -dsnd -u8 -f

RtpAudio -smic -d10.0.1.62:16384 -u8 -f
```

Notes:

- Above IP addresses and UDP ports are only examples. These values may be different from one test system to another.

 Copyright © Signalogic 2012-2015

- Make sure the WinXP machine IP address matches one of the remote IP addresses previously entered (in step 9 above).

- Only one IP call can be connected to a WinXP machine at one time. This is a limitation of the RTP Audio test program. To test multiple IP calls, use more than one WinXP machine.

- For detailed information about the RTP Audio test program, please see the document "RTP Audio Test Program Users Guide".

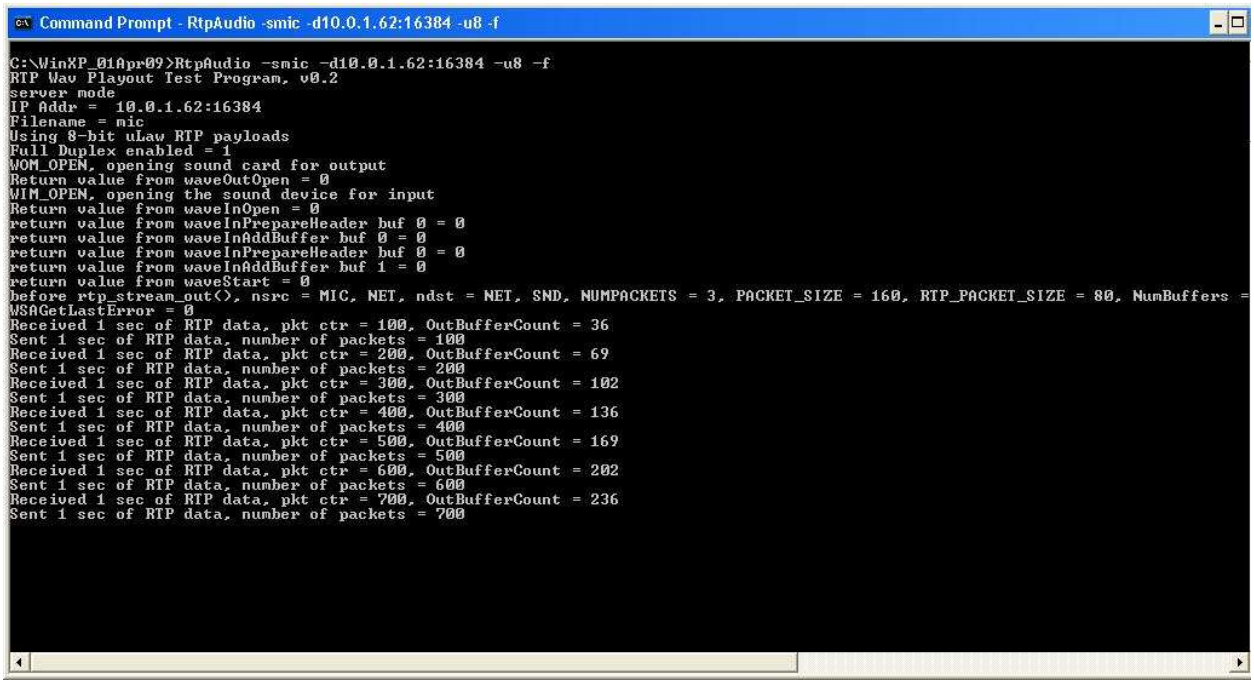

Command Prompt - RtpAudio -smic -d10.0.1.62:16384 -u8 -f
C:\WinXP_01Apr09>RtpAudio -smic -d10.0.1.62:16384 -u8 -f
RTP Wav Playout Test Program, v0.2
server mode
IP Addr = 10.0.1.62:16384
Filename = mic
Using 8-bit uLaw RTP payloads
Full Duplex enabled = 1
WOM_OPEN, opening sound card for output
Return value from waveOutOpen = 0
WIM_OPEN, opening the sound device for input
Return value from waveInOpen = 0
return value from waveInPrepareHeader buf 0 = 0
return value from waveInAddBuffer buf 0 = 0
return value from waveInPrepareHeader buf 0 = 0
return value from waveInAddBuffer buf 1 = 0
return value from waveStart = 0
before rtp_stream_out(), nsrc = MIC, NET, ndst = NET, SND, NUMPACKETS = 3, PACKET_SIZE = 160, RTP_PACKET_SIZE = 80, NumBuffers =
WSAGetLastError = 0
Received 1 sec of RTP data, pkt ctr = 100, OutBufferCount = 36
Sent 1 sec of RTP data, number of packets = 100
Received 1 sec of RTP data, pkt ctr = 200, OutBufferCount = 69
Sent 1 sec of RTP data, number of packets = 200
Received 1 sec of RTP data, pkt ctr = 300, OutBufferCount = 102
Sent 1 sec of RTP data, number of packets = 300
Received 1 sec of RTP data, pkt ctr = 400, OutBufferCount = 136
Sent 1 sec of RTP data, number of packets = 400
Received 1 sec of RTP data, pkt ctr = 500, OutBufferCount = 169
Sent 1 sec of RTP data, number of packets = 500
Received 1 sec of RTP data, pkt ctr = 600, OutBufferCount = 202
Sent 1 sec of RTP data, number of packets = 600
Received 1 sec of RTP data, pkt ctr = 700, OutBufferCount = 236
Sent 1 sec of RTP data, number of packets = 700

Figure 4-24, Full-duplex IP call testing using RTP Audio test program

14. Verify effects of AGC by pressing the 'a' key (Algorithm Mode) and then the 'g' key several times and noting slight difference in voice levels and background noise. When you do this you should see a display similar to Figure 4-25 below:

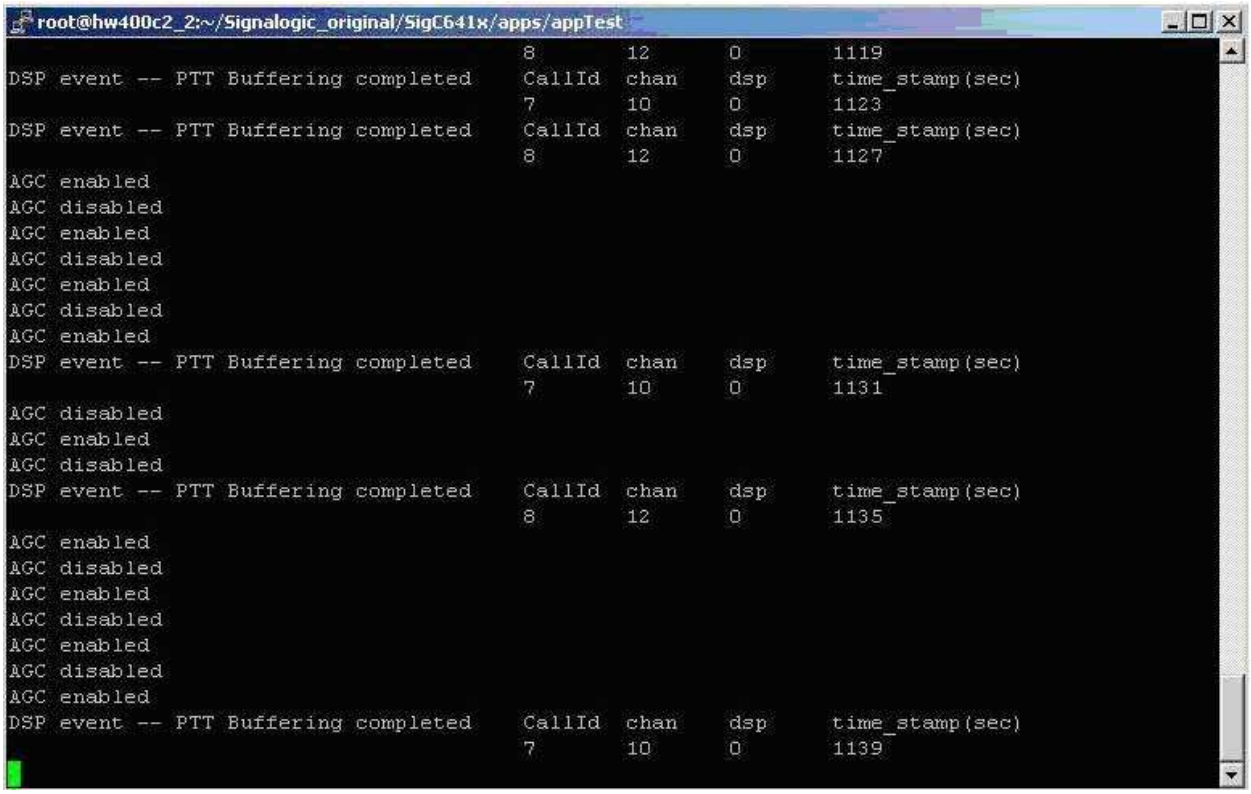


Figure 4-25, Display after entering Algorithm Mode and toggling AGC Active

If a digital scope is connected as shown in "Notes and Equipment" section above, note that effects of AGC are visible on the scope. If AGC is not active, then the output trace (green) will show a flat line (zero pattern) for any unused channels. If AGC is active, then the output trace will show a constant, non-zero pattern for any unused channels. (Remember that if N calls are set up, then unused channels start after 2*N channels on the scope display).

15. Exit Algorithm Mode (press 'a' key again) and then press the 'd' key to display a "diagnostic readout" showing current call state information maintained by the target code. You should see a display similar to Figure 4-26 below:

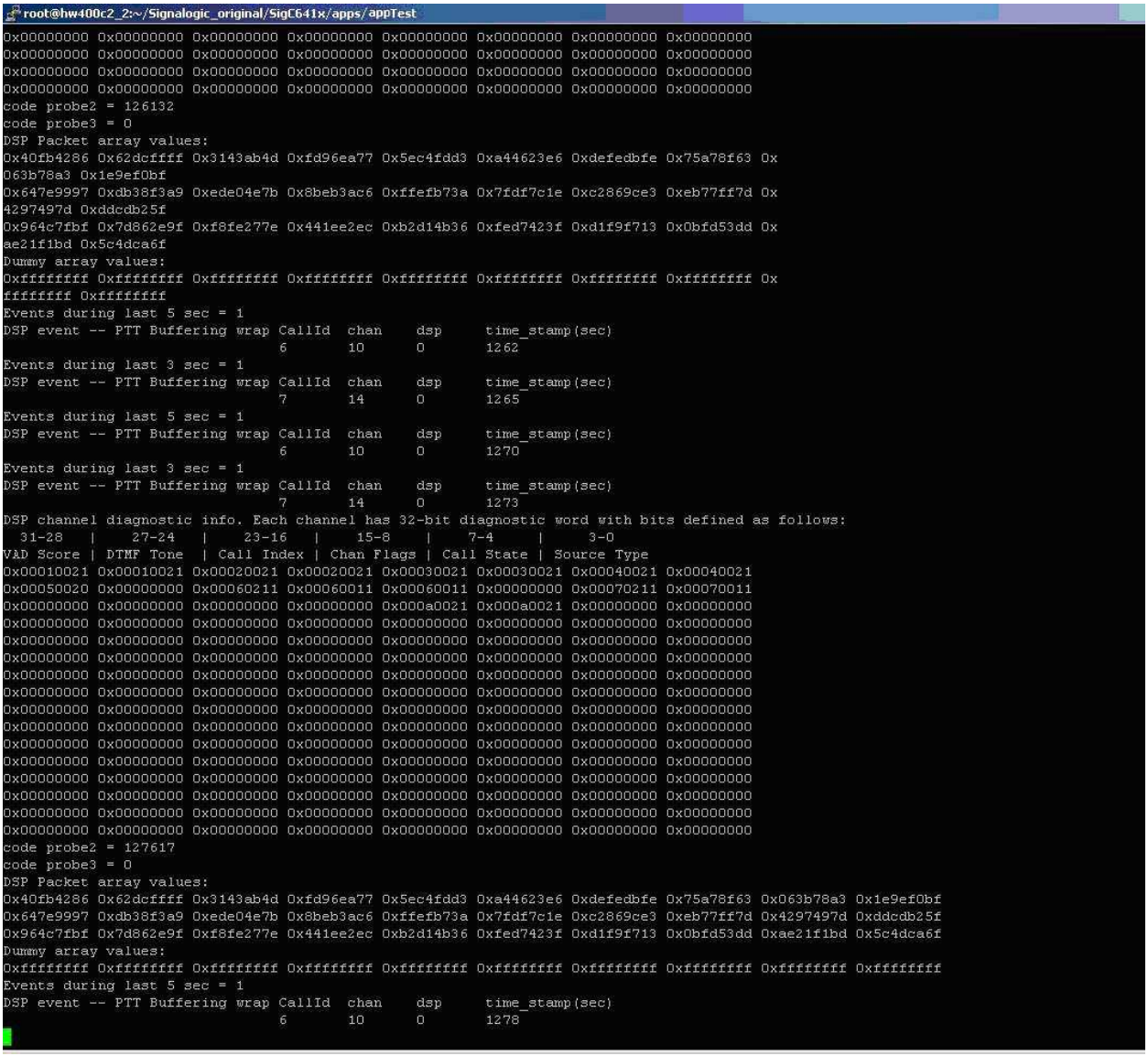


Figure 4-26, Diagnostic data display

The diagnostic data display includes target code “probes” (used for monitoring certain sections of target code), IP call transmit (send) packet values (RTP payload only), and a target channel table, which shows a 32-bit encoded “channel information” value for each channel (channels 0 to 127). Channel information values have the following format:

31-28	27-24	23-16	15-8	7-4	3-0
VAD Score	DTMF Detect	Call Index	Channel Flags	Call State	Channel Source

Encoded field values are explained in the table below:

 Copyright © Signalogic 2012-2015

Field	**Description**	**Encoded Values**
VAD Score	Current VAD score for the channel	0-15, with increasing score indicating higher confidence that voice has been detected
DTMF Detect	Most recent DTMF tone detected on the channel	0-15, corresponding to touch pad keys (or digits) on the handset as follows: **Encoded Value** — **Key or Digit** 0 — 1 1 — 2 2 — 3 3 — A [1] 4 — 4 5 — 5 6 — 6 7 — B [1] 8 — 7 9 — 8 10 — 9 11 — C [1] 12 — * 13 — 0 14 — # 15 — D [1]
Call Index	Index of call to which this channel is connected. A call can have multiple nodes, so an index value of N indicates "this channel is a node of call N".	1-255
Channel Flags	Indicators for activity on the channel. Note that flags use "power of 2" encoding as more than one channel flag can be active at any one time	1 Indicates channel is a PTT listener 2 Indicates channel is a PTT talker 4 Indicates a media playout is currently active on the channel 8 Indicates a DTMF tone has been detected on the channel
Call State	Current state of the call to which this channel is connected.	0 Loopback 1 PTT Call 2 2-Party Call 3 Conference Call

 Copyright © Signalogic 2012-2015

Channel Source	Voice / data source for the channel	0	None
		1	TDM (CTBus, McBSP0) [2]
		2	IP [3]
		3	H.110 (McBSP1)
		4	Inter-DSP (McBSP1)

Notes:

[1] If handset has a fourth (4th) column of keys.

[2] A TDM channel connects to one of 28 T1 streams on the DS3/T3 module.

[3] In "basic IP" mode, IP channels are processed by the host CPU; i.e. the appTest program reads/writes IP/UDP/RTP packets from target CPU memory, and sends/receives the packets over the network. In "Advanced IP" mode, IP channels are processed by target CPU code; i.e. target CPU code sends/receives IP/UDP/RTP packets using the GbE interface on the SigC641x module. In both cases the receive-side jitter buffer is implemented in target CPU code and memory.

 Copyright © Signalogic 2012-2015

16. Press keypad keys on a connected handset and verify that you see DTMF event display as shown in Figure 4-27 below.

```
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        0       0       0       676
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        1       0       0       686
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        8       1       0       689
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        0       1       0       692
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        6       0       0       695
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        1       1       0       703
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        3       0       0       704
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        2       1       0       705
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        2       0       0       707
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        8       0       0       708
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        6       1       0       709
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        5       1       0       710
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        *       1       0       712
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        *       0       0       714
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        9       0       0       716
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        7       0       0       718
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        3       0       0       719
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        2       1       0       722
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        8       1       0       724
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
                        9       1       0       725
DSP event -- DTMF       tone    chan    dsp     time_stamp(sec)
```

Figure 4-27, DTMF event display

Note that DTMF events are detected only on channels currently allocated for calls; i.e. if a channel is not connected to a phone, then no DTMF events are detected or displayed.

17. Press the ‘L’ key to load one or more medias. For example load in_begin_low.wav as first media, test.wav as second media, test2_mu.wav as third media, etc. The following waveform file formats are supported:

- .wav, PCM (16-bit linear)
- .wav u-Law (8-bit)
- .wav A-Law (8-bit)
- .tim, PCM (16-bit linear)
- .ton, PCM (16-bit linear after de-compression)

Note that only mono (single-channel) waveform files are supported.

18. Press ‘m’ key to show a list of loaded medias, as shown in Figure 4-28 below.

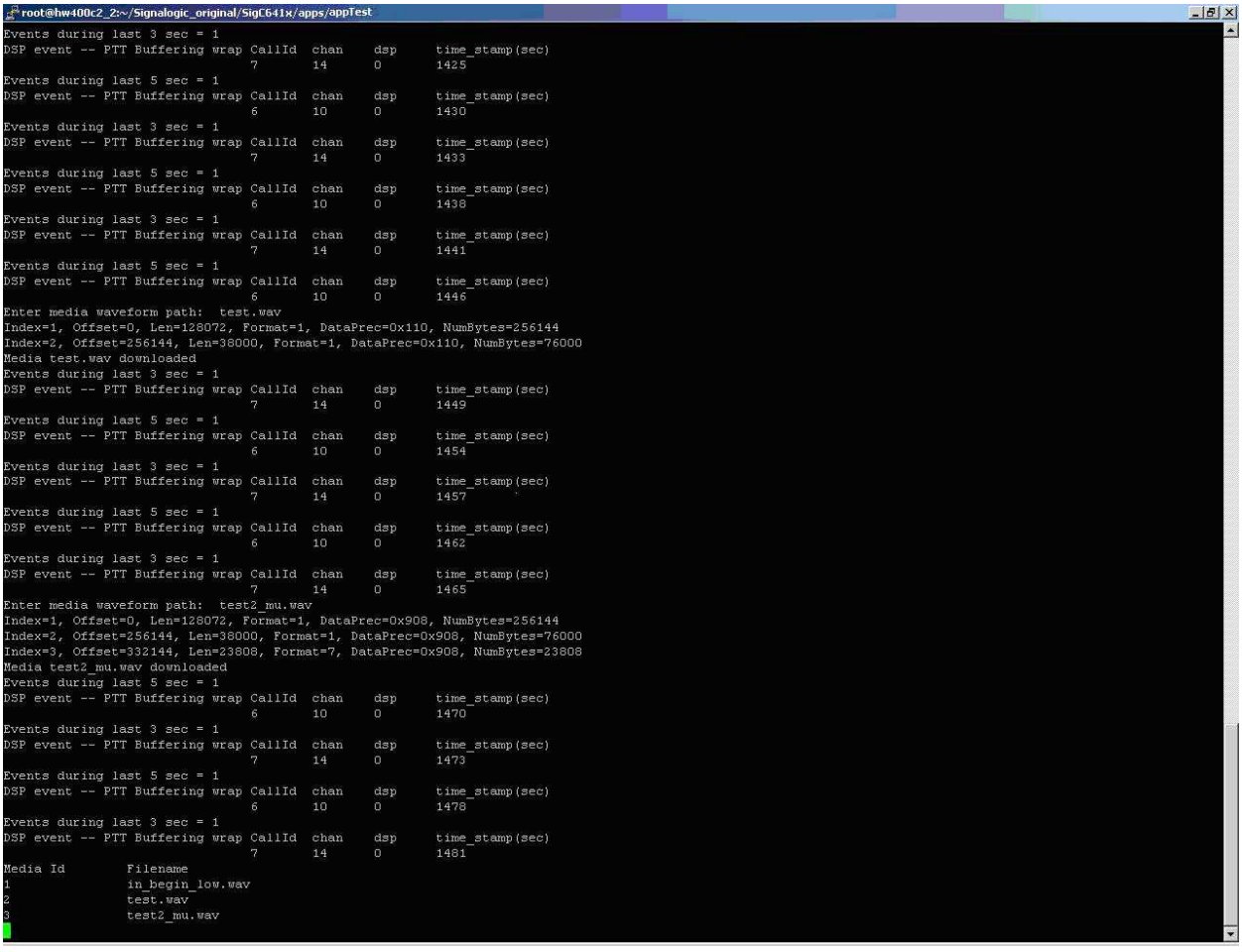

root@hw400c2_2:~/Signalogic_original/SigC641x/apps/appTest
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1425
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1430
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1433
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1438
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1441
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1446
Enter media waveform path: test.wav
Index=1, Offset=0, Len=128072, Format=1, DataPrec=0x110, NumBytes=256144
Index=2, Offset=256144, Len=38000, Format=1, DataPrec=0x110, NumBytes=76000
Media test.wav downloaded
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1449
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1454
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1457
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1462
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1465
Enter media waveform path: test2_mu.wav
Index=1, Offset=0, Len=128072, Format=1, DataPrec=0x908, NumBytes=256144
Index=2, Offset=256144, Len=38000, Format=1, DataPrec=0x908, NumBytes=76000
Index=3, Offset=332144, Len=23808, Format=7, DataPrec=0x908, NumBytes=23808
Media test2_mu.wav downloaded
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1470
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1473
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1478
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1481
Media Id Filename
1 in_begin_low.wav
2 test.wav
3 test2_mu.wav

Figure 4-28, Example list of loaded Medias

19. Press ‘p’ key to play medias. The media playout prompt entry is of the format:

```
id, rc, ch
```

Where id is Media Id, rc is Repeat Count, and ch is Channel on which to play the media. For example, to play Media Id 1 on channel 0, enter at the playout prompt:

```
1,1,0
```

To play Media Id 2 on channel 1, enter:

```
2,3,1
```

20. To unload (delete) a media, press the ‘u’ key, and enter the Media Id to be deleted.

4.4.5.6.1 Media Playout Notes

- A “media playout completed” event is displayed when each media playout is complete. An example is given in Figure 4-29 below
- If a media playout is attempted on a channel already playing a media, then an error message is displayed.
- If a media is played on a *channel not used* in a currently active call, then a *temporary* T8110 channel connection is made, in only one direction, to allow the media playout to be audible

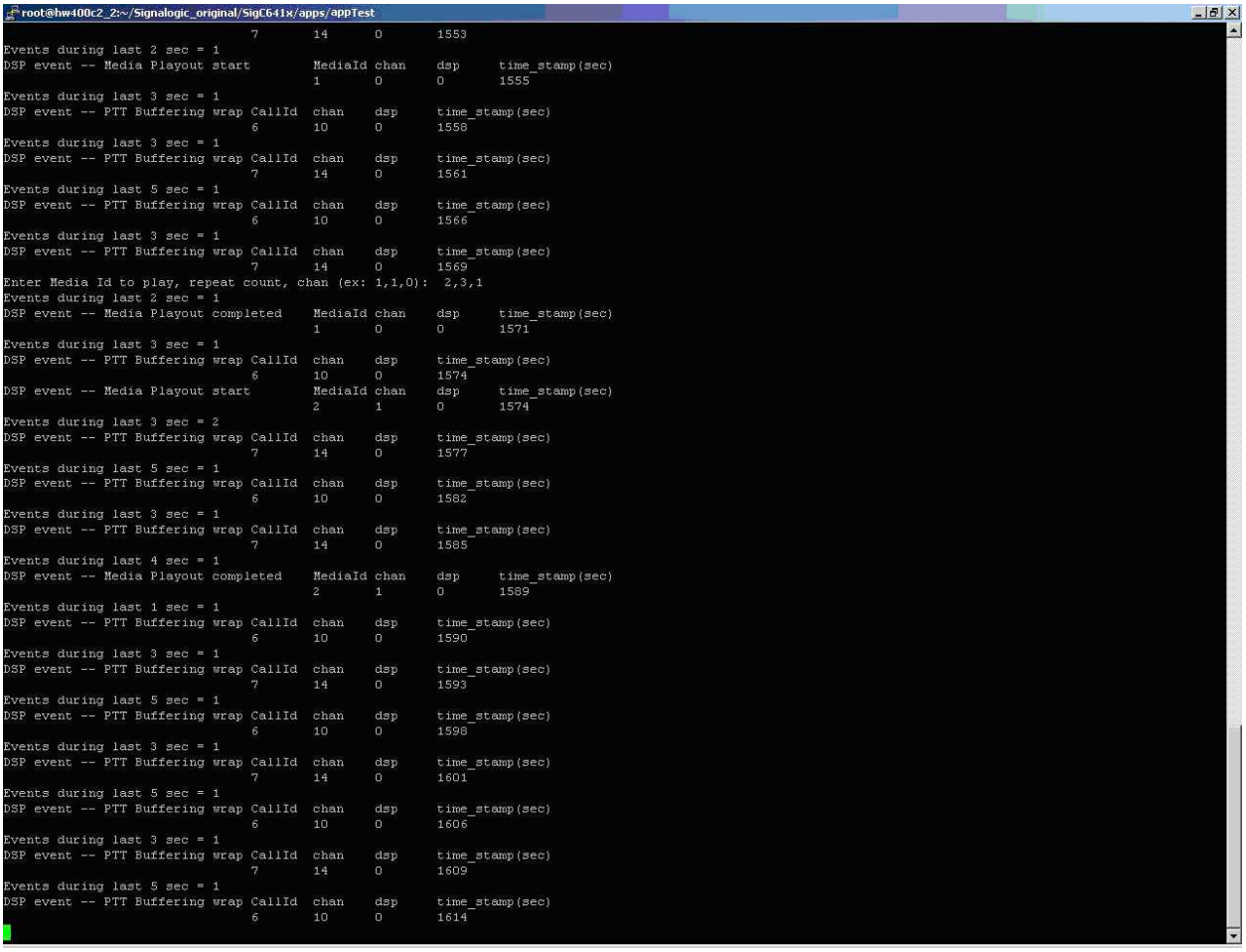

root@hw400c2_2:~/Signalogic_original/SigC641x/apps/appTest
7 14 0 1553
Events during last 2 sec = 1
DSP event -- Media Playout start MediaId chan dsp time_stamp(sec)
1 0 0 1555
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1558
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1561
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1566
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1569
Enter Media Id to play, repeat count, chan (ex: 1,1,0): 2,3,1
Events during last 2 sec = 1
DSP event -- Media Playout completed MediaId chan dsp time_stamp(sec)
1 0 0 1571
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1574
DSP event -- Media Playout start MediaId chan dsp time_stamp(sec)
2 1 0 1574
Events during last 3 sec = 2
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1577
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1582
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1585
Events during last 4 sec = 1
DSP event -- Media Playout completed MediaId chan dsp time_stamp(sec)
2 1 0 1589
Events during last 1 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1590
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1593
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1598
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1601
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1606
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1609
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1614

Figure 4-29, Media playout event display

21. When media playout is finished, use the handsets to verify calls continue to function correctly.

22. Measuring CPU usage, open another terminal window (use putty again) and connect to the HW400c/2 board. After you login and get a command-line prompt, enter the command '**top**'. You should see the CPU usage of different processes, shown as a percentage. Figure 4-30 below shows CPU usage by the appTest process as varying from 3 to 4%.

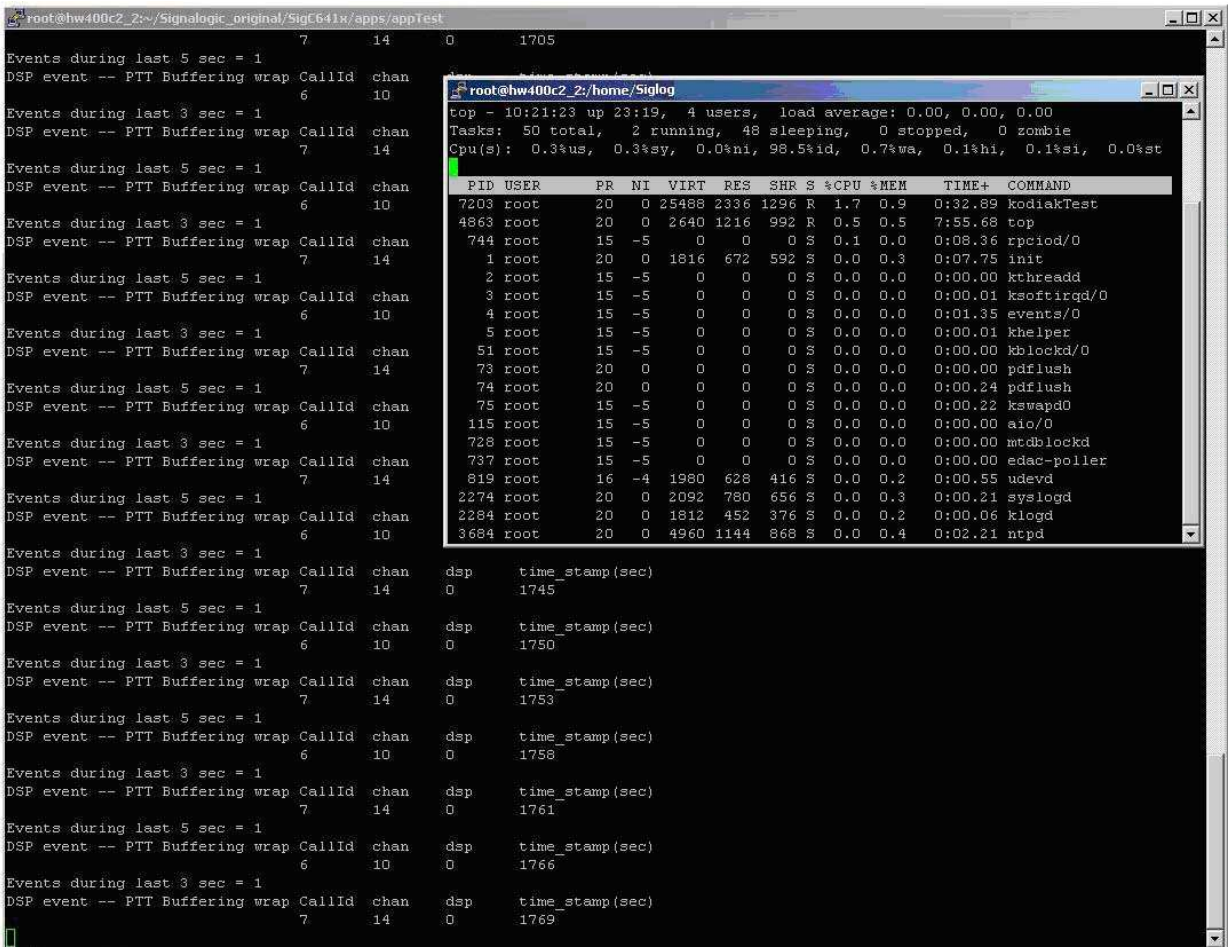

root@hw400c2_2:~/Signalogic_original/SigC641x/apps/appTest
root@hw400c2_2:/home/Siglog
top - 10:21:23 up 23:19, 4 users, load average: 0.00, 0.00, 0.00
Tasks: 50 total, 2 running, 48 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.3%us, 0.3%sy, 0.0%ni, 98.5%id, 0.7%wa, 0.1%hi, 0.1%si, 0.0%st
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
7203 root 20 0 25488 2336 1296 R 1.7 0.9 0:32.89 kodiakTest
4863 root 20 0 2640 1216 992 R 0.5 0.5 7:55.68 top
744 root 15 -5 0 0 0 S 0.1 0.0 0:08.36 rpciod/0
1 root 20 0 1816 672 592 S 0.0 0.3 0:07.75 init
2 root 15 -5 0 0 0 S 0.0 0.0 0:00.00 kthreadd
3 root 15 -5 0 0 0 S 0.0 0.0 0:00.01 ksoftirqd/0
4 root 15 -5 0 0 0 S 0.0 0.0 0:01.35 events/0
5 root 15 -5 0 0 0 S 0.0 0.0 0:00.01 khelper
51 root 15 -5 0 0 0 S 0.0 0.0 0:00.00 kblockd/0
73 root 20 0 0 0 0 S 0.0 0.0 0:00.00 pdflush
74 root 20 0 0 0 0 S 0.0 0.0 0:00.24 pdflush
75 root 15 -5 0 0 0 S 0.0 0.0 0:00.22 kswapd0
115 root 15 -5 0 0 0 S 0.0 0.0 0:00.00 aio/0
728 root 15 -5 0 0 0 S 0.0 0.0 0:00.00 mtdblockd
737 root 15 -5 0 0 0 S 0.0 0.0 0:00.00 edac-poller
819 root 16 -4 1980 628 416 S 0.0 0.2 0:00.55 udevd
2274 root 20 0 2092 780 656 S 0.0 0.3 0:00.21 syslogd
2284 root 20 0 1812 452 376 S 0.0 0.2 0:00.06 klogd
3684 root 20 0 4960 1144 868 S 0.0 0.4 0:02.21 ntpd
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1745
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1750
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1753
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1758
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1761
Events during last 5 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
6 10 0 1766
Events during last 3 sec = 1
DSP event -- PTT Buffering wrap CallId chan dsp time_stamp(sec)
7 14 0 1769

Figure 4-30, Example of running Linux Top process

23. Use ‘t’ key to tear down calls 1-7 and call 10. Note effects on scope if connected. Verify using handsets that no calls are active (per-Channel loopback state is restored).

24. Press ‘c’ key again to verify no calls are active. Figure 4-31 shows the call list after the calls are tear down.

 Copyright © Signalogic 2012-2015

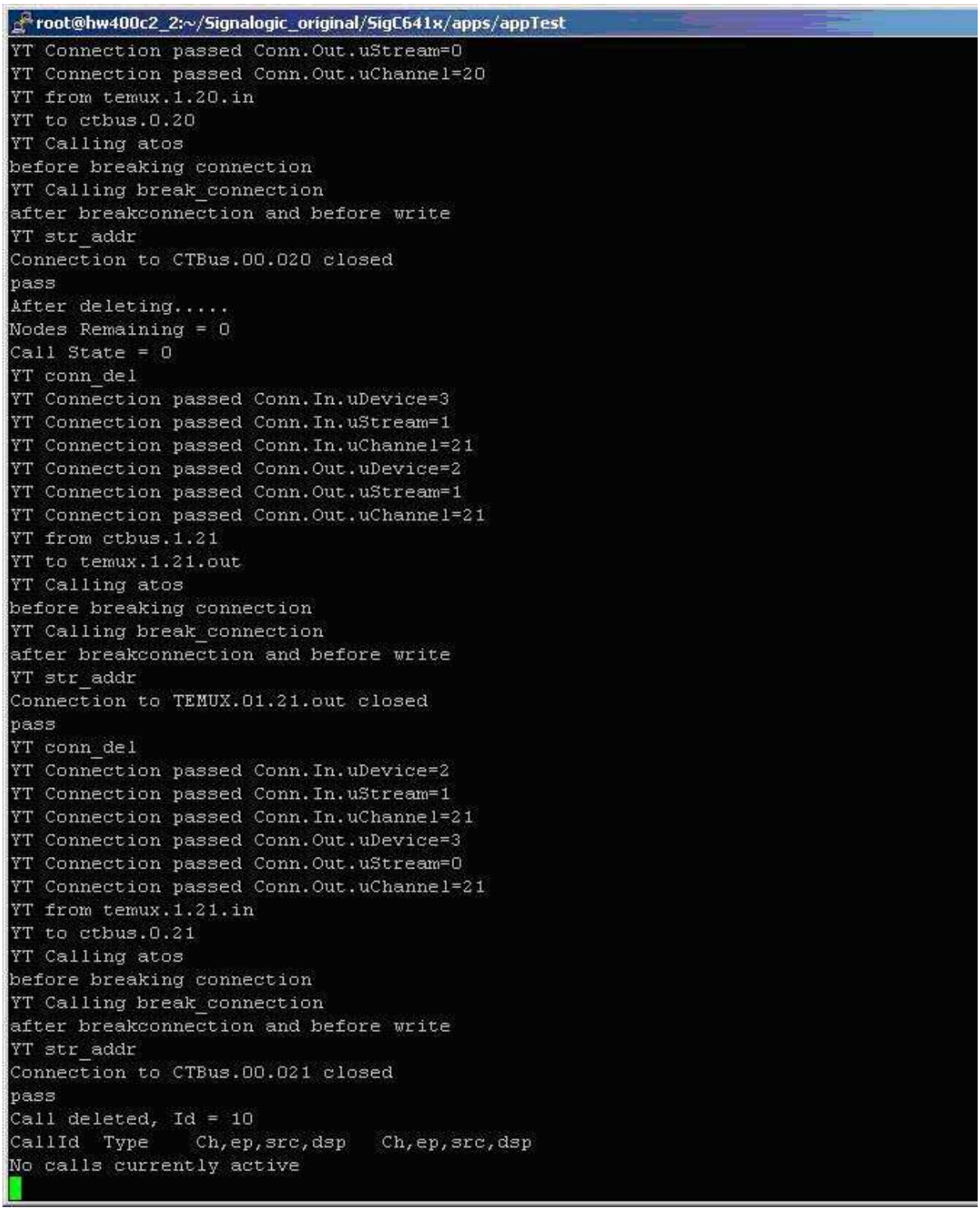


Figure 4-31, Call list after call tear down

25. Optional Lab equipment Verification. If a digital scope is connected to the CTBus between the DS3 module and SigC641x modules, then the scope should show a display similar to Figure 4-32 below.

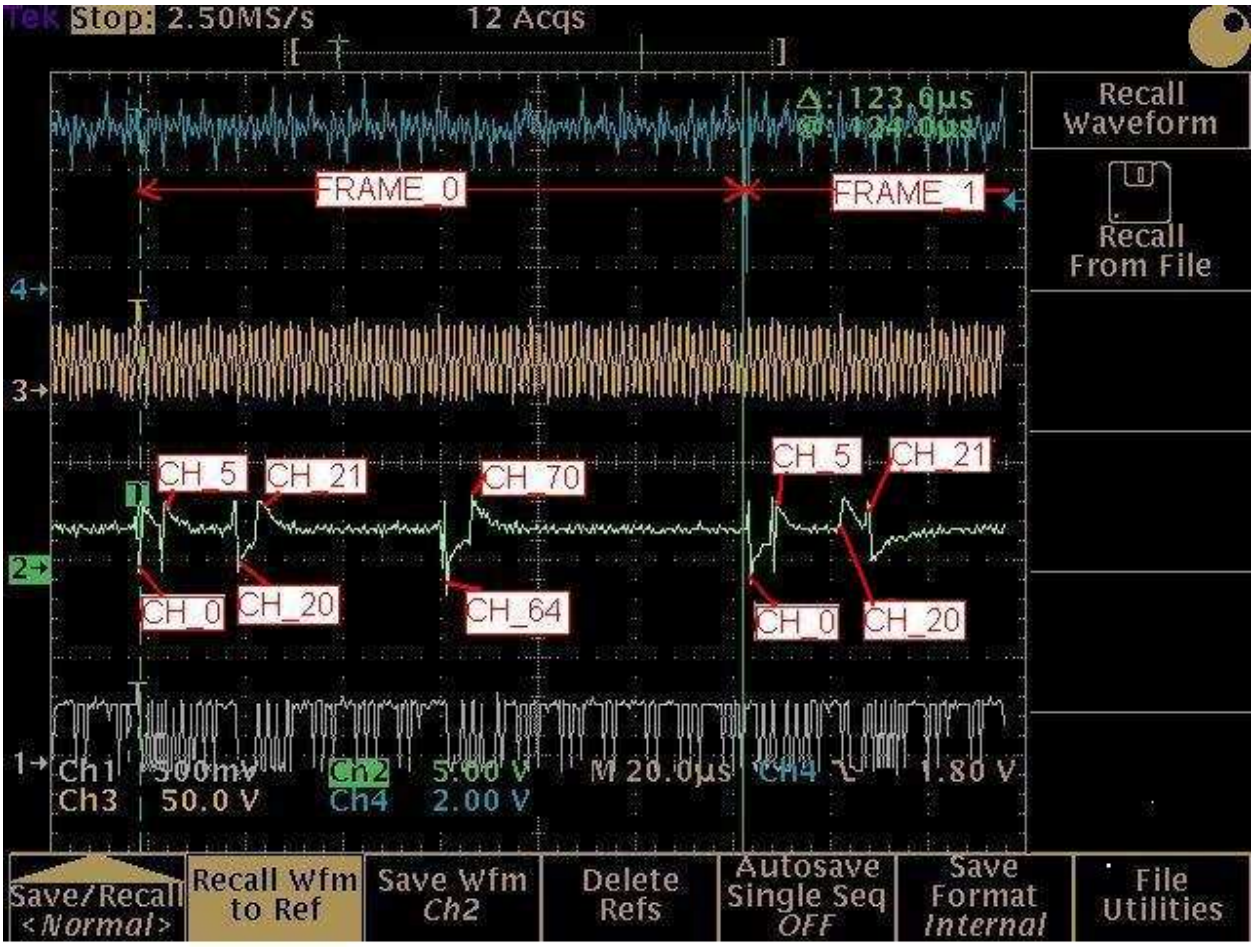


Figure 4-32, Capture of active CTBus data between SigC641x and DS3 modules

26. When necessary, press the ‘q’ key to exit the appTest program.

4.5 Texas Instruments Tools Installation

***Note 1:** This section covers installation of TI Linux command-line tools, which are required for SigC667x systems. For SigC641x systems, TI Code Composer Studio installation is required on a WinXP or Win7/8 system -- see the SigC5x / SigC6x Hardware Reference Guide for more information.*

***Note 2:** If during installation of TI files in ".bin" format you get what looks like an "immediate abort", i.e. the install doesn't appear to do or print anything and bounces you back to the command line, you may need to first run one or both of the following apps:*

```
apt-get install ia32-libs  (for example if you're running Ubuntu 12.xx)

apt-get install libc6:i386 (for example if you're running Ubuntu 14.xx)

yum install compat-libstdc++-33.i686 (for example if you're running CentOS7)
```

This issue can arise since the TI installer doesn't appear to work for some 64-bit Linux distributions, but it will work with correct 32-bit libs installed.

4.5.1 TI Code Generation Tools Download

The first step is to download TI Code Generation tools (commonly referred to as CG Tools). Go to the following page:

https://www-a.ti.com/downloads/sds_support/TICodegenerationTools/download.htm

Note that this page may require a log-in; if so you have to register with TI for an account (if you don't already have one). For more information, go to this TI page:

http://processors.wiki.ti.com/index.php/Compiler_Releases

Notes about the above page:

1. If you click under "Free Compiler Downloads" (on link labeled "Here"), it will take you to the CG Tools download page (first link above).

2. You do not need to have Code Composer Studio (CCS) installed.

After you are logged-in, you should see a CG Tools download page similar to Figure 4-33 below.

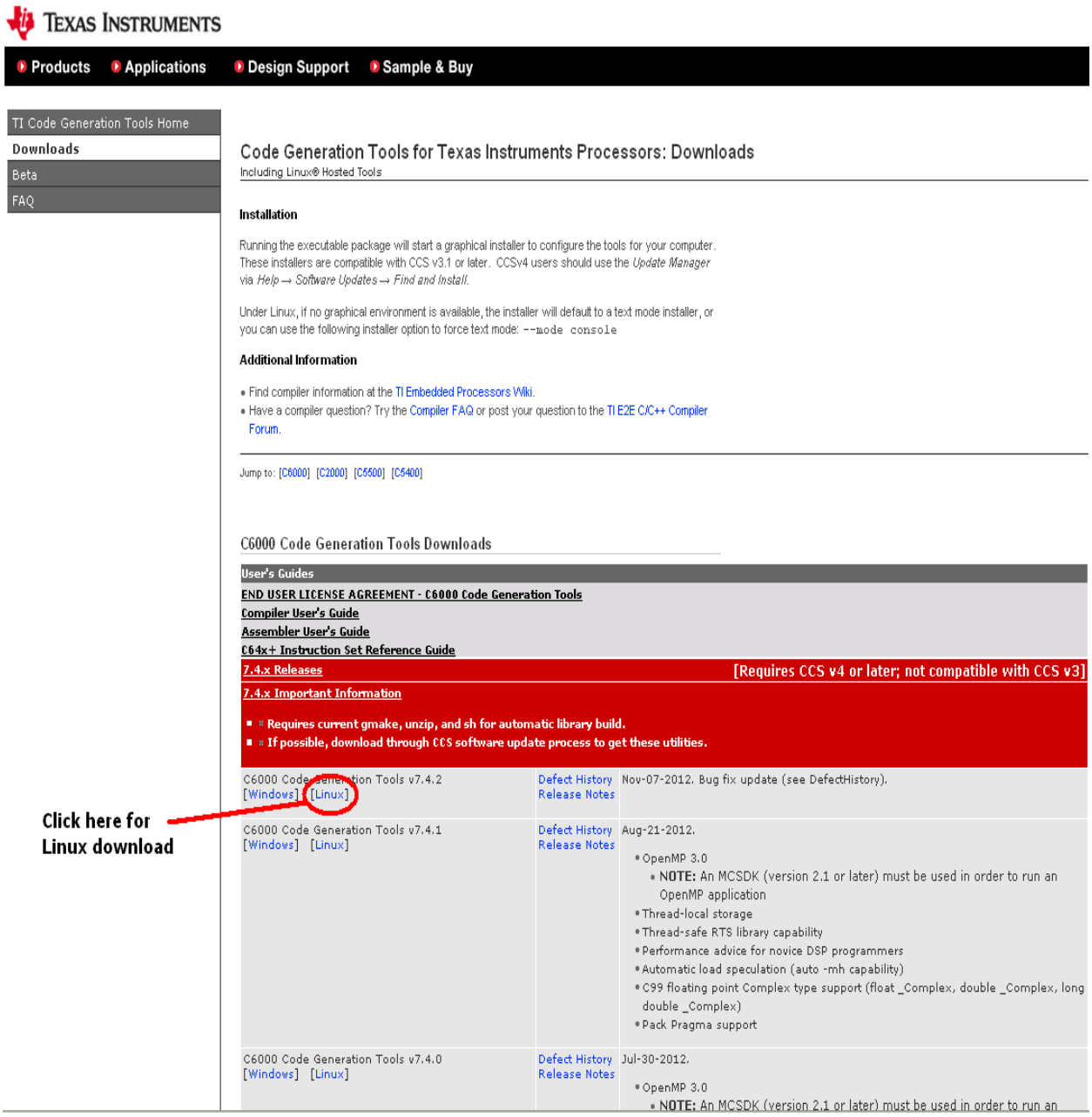

Texas Instruments
Products
Applications
Design Support
Sample & Buy
TI Code Generation Tools Home
Downloads
Beta
FAQ
Code Generation Tools for Texas Instruments Processors: Downloads
Including Linux® Hosted Tools
Installation
Running the executable package will start a graphical installer to configure the tools for your computer. These installers are compatible with CCS v3.1 or later. CCSv4 users should use the Update Manager via Help → Software Updates → Find and Install.
Under Linux, if no graphical environment is available, the installer will default to a text mode installer, or you can use the following installer option to force text mode: --mode console
Additional Information
• Find compiler information at the TI Embedded Processors Wiki.
• Have a compiler question? Try the Compiler FAQ or post your question to the TI E2E C/C++ Compiler Forum.
Jump to: [C6000] [C2000] [C5500] [C5400]
C6000 Code Generation Tools Downloads
User's Guides
END USER LICENSE AGREEMENT - C6000 Code Generation Tools
Compiler User's Guide
Assembler User's Guide
C64x+ Instruction Set Reference Guide
7.4.x Releases
[Requires CCS v4 or later; not compatible with CCS v3]
7.4.x Important Information
Requires current gmake, unzip, and sh for automatic library build.
If possible, download through CCS software update process to get these utilities.
C6000 Code Generation Tools v7.4.2
[Windows] [Linux]
Defect History
Release Notes
Nov-07-2012. Bug fix update (see DefectHistory).
C6000 Code Generation Tools v7.4.1
[Windows] [Linux]
Defect History
Release Notes
Aug-21-2012.
• OpenMP 3.0
• NOTE: An MCSDK (version 2.1 or later) must be used in order to run an OpenMP application
• Thread-local storage
• Thread-safe RTS library capability
• Performance advice for novice DSP programmers
• Automatic load speculation (auto -mh capability)
• C99 floating point Complex type support (float _Complex, double _Complex, long double _Complex)
• Pack Pragma support
C6000 Code Generation Tools v7.4.0
[Windows] [Linux]
Defect History
Release Notes
Jul-30-2012.
• OpenMP 3.0
• NOTE: An MCSDK (version 2.1 or later) must be used in order to run an
Click here for Linux download

Figure 4-33, TI Code Generation Tools download page

Download the following TI files to /home/<user>/Downloads:

ti_cgt_c6000_8.0.3_linux_installer_x86.bin
bios_mcsdk_02_01_02_06_setuplinux.bin

```
cd /home/<user>/Downloads
chmod 777 ti_cgt_c6000_8.0.3_linux_installer_x86.bin
chmod 777 bios_mcsdk_02_01_02_06_setuplinux.bin
mkdir /opt/ti     (you will be prompted for a path to install, use this path)
```

 Copyright © Signalogic 2012-2015

```
./bios_mcsdk_02_01_02_06_setuplinux.bin
./ti_cgt_c6000_8.0.3_linux_installer_x86.bin
```

Install into the /opt/ti/ directory when prompted for install path.

4.5.2 TI RTOS-MCSDK Download

The second step is to download TI's TI-RTOS-MCSDK, which will download the following components:

- SYSBIOS (Real-time operating system for TI multicore CPUs)
- IPC (Interprocessor Communication)
- XDCtools (utilities and libraries)
- PDK (Platform Development Kit)

To download the TI-RTOS-MCSDK, go to the following page:

http://software-dl.ti.com/sdoemb/sdoemb_public_sw/bios_mcsdk/latest/index_FDS.html

and locate the product download link with the following label:

```
bios_mcsdk_02_01_02_06_setuplinux.bin
```

and click on that to start the download. Note that the version number portion of the label will change over time (in the link above, this is v2.01.02.06).

Notes about SYSBIOS, IPC, and XDCtools:

1. The version of SYSBIOS installed should be 6.32.05.54 or higher. The version of XDCtools installed should be 3.22.04.46 or higher. These version numbers have been tested in Signalogic labs, although older versions may still work.

2. IPC (Interprocessor Communication) is currently not used by DirectCore and CIM software, but may be in the future. For C667x devices, IPC includes SRIO and HyperLink. The SigC6678 cards have SRIO connections between all C667x devices. SRIO is a fast way to move data between C667x processors. HyperLink is also fast, but connects only pairs of devices together.

3. PDK (Platform Development Kit) is installed with TI-RTOS-MCSDK. PDK is currently used by DirectCore and CIM software. Also, PDK includes TI' CSL (Chip Support Library), which is not used currently DirectCore and CIM software, but is likely to be used in the future.

After downloading and installing TI tools as described in preceding sections, you should have a TI software folder hierarchy similar to that shown in Figure 4-34 below.

Note: The default location for TI software is the /opt/ti folder, at the same level as /root, /home, etc.

 Copyright © Signalogic 2012-2015

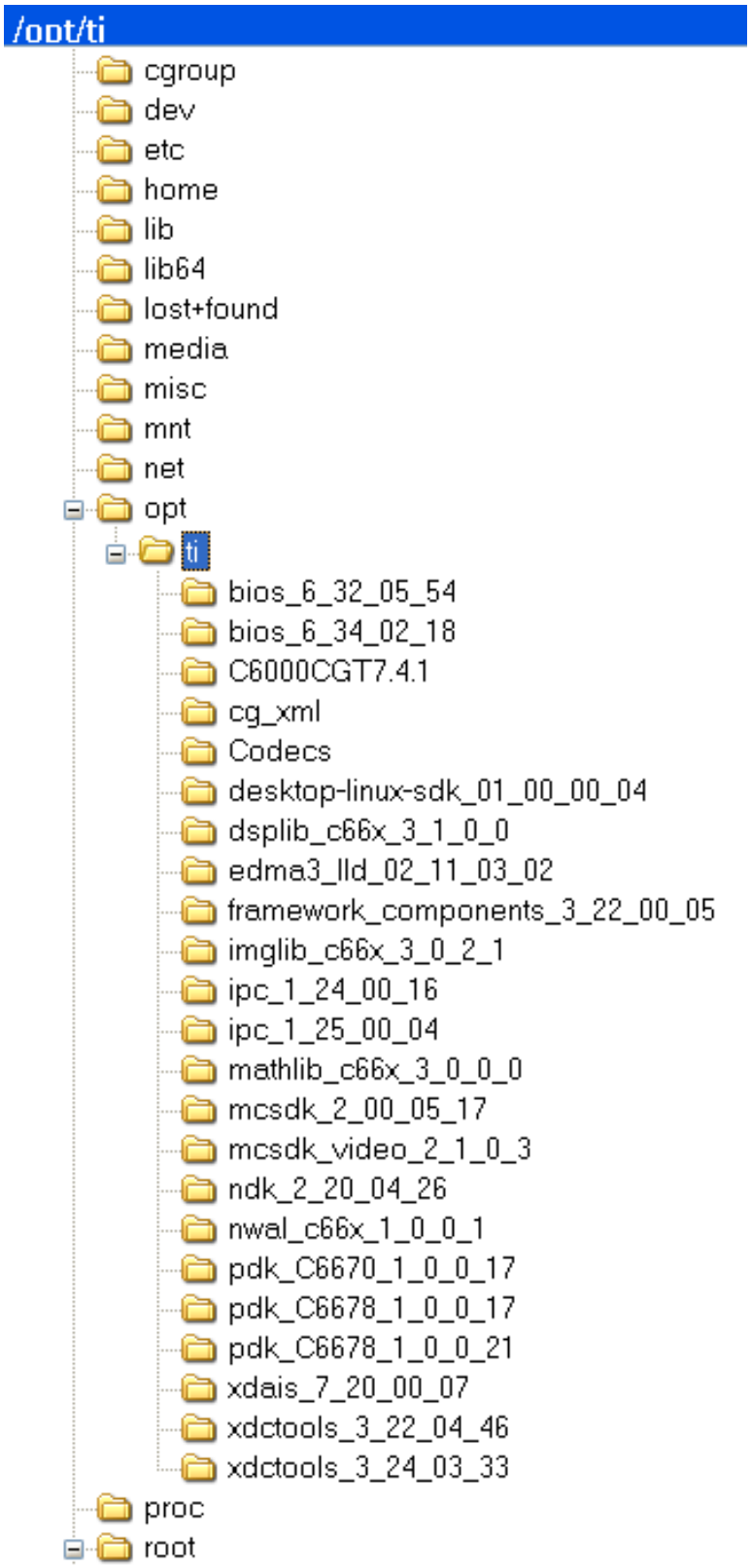


Figure 4-34, TI software folder hierarchy, including Code Gen tools, SYSBIOS, IPC, XDCtools, and PDK

4.5.3 TI SYSBIOS, IPC, and XDCtools Download

If needed, SYSBIOS, IPC, and XDCtools can be downloaded separately, for example to upgrade one of the components without downloading all of TI-RTOS-MCSDK. To download components separately, go to the following page:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/bios/sysbios/index.html

Note that currently, PDK cannot be downloaded separately, and must be obtained via TI-RTOS-MCSDK download.

 Copyright © Signalogic 2012-2015

4.6 RTAF Software Usage

This section covers use of the Real-Time Algorithm Framework (RTAF) software. RTAF software includes the following components and run-time functionality:

- for TI targets, SYSBIOS configuration, including run-time initialization and memory allocation, timer configuration (OS tick, profile timers, etc)
- CSL, NDK, and other library and initialization
- Run-time initialization, including CPU boot
- Peripheral drivers and interrupt service routines (both hardware and software interrupts)
- Utility functions
- Algorithm functions
- DirectCore properties used for host communication with certain utility and algorithm functions
- CIM software support

4.6.1 RTAF Configuration Options

The RTAF software build process supports the following pre-processor definitions

Pre-Processor Definition	Description
_USE_CM_	Enables Call Manager support
_PERIPH_TEST_	Enables POST and other test options for peripherals
BIOS	Enables SYSBIOS compatibility in the build process
_CIMF	Enables support required for CIM software builds
_ANALOG_IO_	Enables analog I/O support on SigC641x modules (default peripheral interface is McBSP0)
TDM	Enables TDM support on SigC641x modules (default peripheral interface is McBSP1)
_H110_	Enables H.110 support on SigC641x modules (default peripheral interface is McBSP2)

_USE_PN4_	Enables PN4 connector interface on SigC641x modules

Pre-processor definitions can be set in the CCS project (under Project main menu item, Build Options, Compiler tab), or in Linux command-line build / make files.

4.6.2 RTAF Peripheral Support

RTAF supports the following C64x / C66x peripherals:

Peripheral	Comments
Host (HPI or PCIe)	HPI32 is supported on SigC641x, PCIe is supported on SigC667x
Ethernet	If present on the hardware. In the case of SigC667x, the oncard 1 GbE interface is supported, using TI's NDK and PA (Packet Accelerator) interfaces
McBSP	For SigC641x, supports audio/analog I/O, TDM, H.110, and user-defined interfaces. Not currently supported on SigC667x (Telecom Serial Port, or TSIP, may be supported at a future point)
GPIO	Supports both SigC641x and SigC667x
Flash devices	Typically connected to EMIFB interface
FPGA Logic	If present on the card, for example host / CTBus FPGA on SigC641x modules
UART	If present on the hardware
RS-422	If present on the hardware

4.6.3 RTAF Development and Host Platform Notes

As noted in section 4 above, Software Installation, for SigC667x cards, RTAF software is installed under the mCPU_target folder on Linux systems as shown in Figure 4-35a below.

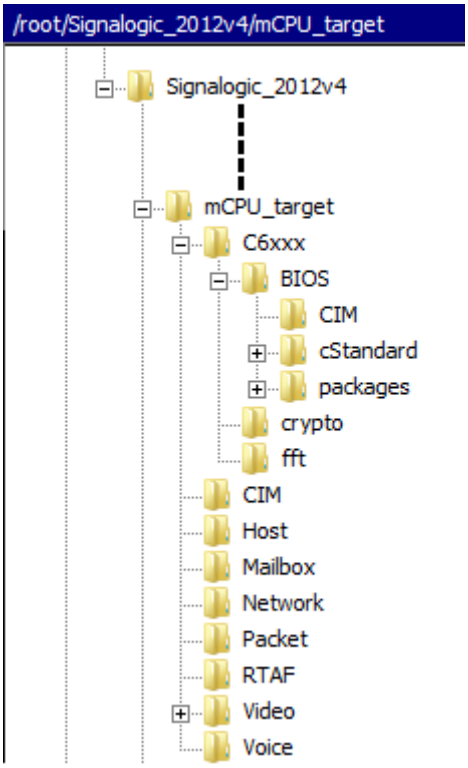


Figure 4-35a, RTAF software folder hierarchy for SigC667x cards

For SigC641x modules on Windows systems, target software is handled differently (as explained in section 4 above, Software Installation, with additional explanation below), and RTAF software is installed as shown in Figure 4-35b below.

Figure 4-35b, target software folder hierarchy for SigC641x modules. RTAF software is included under the C6xxx subfolder

There are a few fundamental points about RTAF development and host platform to keep in mind:

1. RTAF software supports SigC641x modules installed in Linux systems, but *SigC641x RTAF software is not generated or rebuilt on a Linux system*. Instead, the software must be maintained on a WinXP or Win7/8 system with Texas Instruments CCS software installed. Conversely, SigC667x RTAF software builds may be handled using CCS software or Texas Instruments command-line code generation tools (TI's Linux version of CGT). This is one reason why RTAF software installation looks different for C64x vs. C66x based hardware.

2. In Linux systems, RTAF software is stored on the “mCPU_target” subfolder. This naming convention refers to run-time software for multicore CPU or other target hardware; e.g. CIM accelerator, DSP card, video or telecom card, network processing card, neural net card, etc.

4.6.3.1 Linux Platform Builds

In Linux platform builds, there are two (2) important files:

- Makefile
- cimf_build.lst

The cimf_build.lst file is only used if CIM software is being used by host applications, and is located in:

```
/install_path/Signalogic/mCPU_target/C6xxx/TI-RTOS/
```

4.6.3.1.1 Building without CIM

1. First, go to the folder

   ```
   /install_path/Signalogic/mCPU_target/C6xxx/TI-RTOS/cStandard/
   ```

2. Then, use the 'make' command, which invokes the contents of the 'Makefile' file. Note that this file will be overwritten if/when a host-side CIM build script is used.

4.6.3.1.2 Building with CIM

To build with CIM host-side software use the CIM script 'cimpp.sh' located in:

```
/install_path/Signalogic/CIM/cimrt/
```

The command

```
./cimpp.sh -h
```

will display the command line options. An example build command might look like this:

```
./cimpp.sh -s/install_path/Signalogic/CIM/apps/test/ffmpeg_demo/ -px86
-ati66x -cSIGC66XX conv.c
```

The file 'template_Makefile' in that directory can be edited for changes to the build.

Makefile Notes:

1. ORDERED_OBJS includes any libraries (“lib files”) that need to be linked. More libraries can be added as needed.

 Copyright © Signalogic 2012-2015

2. CIM_DEFINES are all the preprocessor macro defines used.
3. CIM_INCLUDES are the include directories for the build process.
4. LINKER_INCLUDES are the include directories for the linker process.

There are 3 targets used in the makefile under the '# All Target' comment. The first one can be edited to change the platform used by changing the -p parameter. The second one builds the source files in the cimf_build.lst file. If CIM software is used by host applications then 2 additional source files will be added to the command to be built. The last tool is the linker, which uses ORDERED_OBJS and all the .obj files in the directory:

```
/install_path/Signalogic/mCPU_target/C6xxx/TI-RTOS/cStandard/
```

4.7 CIM® Hyperpiler™ Software

To install CIM Hyperpiler software unzip the sig_sw_host_cim_vNN.rar file. You should see the following

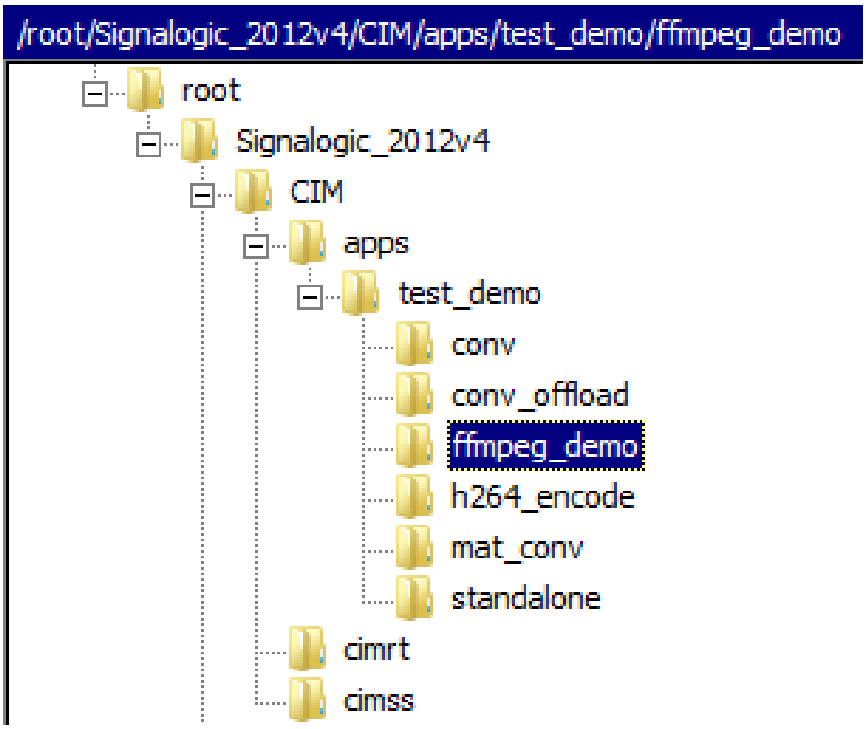


Figure 4-36, CIM host software folder hierarchy, after installation

4.7.1 Running User Programs with CIM

To run user programs with the CIM source code stream generation process, generally follow these guidelines

1. First modify the system PATH environment variable as follows:

```
export PATH=$PATH:/install_path/Signalogic/CIM/cimrt
```

This can be done each time the system boots, or made an automatic boot setting inside the shell profile configuration file, for example /etc/profile, etc.

2. cd (change directory) to the folder with your input C source files.

3. Give a command in the following form to run the CIM pre-processor:

```
cimpp <file1.c> <file2.c> <fileN.c> [-pPlatformType] [-aAccelType] [-cCardType]
```

 Copyright © Signalogic 2012-2015

Where PlatformType, AccelType, and CardType parameters are described in the table below. If no parameters are entered, all output source streams are generated to run on a generic x86 platform with no optimizations. Allowable PlatformType, AccelType, and CardType parameters are listed below:

PlatformType	Description	Support Status
none	x86 + Linux	Y
x86	x86 + Linux	Y
ARM	ARM + Linux	N

AccelType	Description	Support Status
none	Plain Intel x86 code, no optimizations	Y
x86	Intel x86 with OpenMP, IPP, other optimizations	Y
ti64	Texas Instruments C64x	N (Partial, but not supported)
ti66	Texas Instruments C66x	Y

CardType	Description	Support Status
none	If not entered, then default CardType containing CPU matching AccelType is used	Y
SIGC6678	Signalogic 32-core C6678 accelerator	Y
SIGC6678-64	Signalogic 64-core C6678 accelerator	Y

Note that it's also possible to run the CIM pre-processor from an arbitrary location, for example:

```
   cimpp -s/install_path/Signalogic/CIM/apps/test_demo/ffmpeg_demo/ -px86 -ati66x -
cSIGC66XX file1.c file2.c
```

4.7.2 vid_streaming demo

To run the CIM software vid_streaming demo, first cd (change directory) to the vid_streaming folder:

```
   cd /install_path/Signalogic/CIM/apps/test_demo/vid_streaming
```

Then enter:

```
   cimpp -px86 -ati66x -cSIGC66XX vid_streaming.c
```

```
cimpp.sh -px86 -ati66x -cSIGC66XX vid_streaming.c
```

4.7.2.1 vid_streaming CIM Build Description

Figures 4-37a-c below show expected output during the CIM build process for vid_streaming:

```
cimrt$ ./cimpp.sh -s/root/Signalogic_2012v4/CIM/apps/test_demo/ffmpeg_demo/ -px86 -ati66x -cSIGC66XX ffmpeg_demo.c
############################################################
#Running auto build process with following parameters
#
#Main User source file  -  ffmpeg_demo.c
#Number of source files -  1
#Soure file list        -  ffmpeg_demo.c
#Source file path       -  /root/Signalogic_2012v4/CIM/apps/test_demo/ffmpeg_demo/
#Platform Type          -  x86
#Accelerator Type       -  TI66X
#Card Type              -  SIGC66XX
#CCS project directory  -  /root/Signalogic_2012v4/CIM/cimrt/../../mCPU_DSP/C6xxx/BIOS/
#
############################################################
#################
#Invoking Parser#
#################
Executing ./cimss, first input file is ffmpeg_demo.c!
argc = 5
argv = ./cimss -D_LINUX_ -I/root/Signalogic_2012v4/DirectCore/include/ -I/root/Signalogic_2012v4/CIM/apps/test_demo/ffmpeg_demo/ ff
mpeg_demo.c
Before calling frontend
new_argv = ./cimss -D_LINUX_ -I/root/Signalogic_2012v4/DirectCore/include/ -I/root/Signalogic_2012v4/CIM/apps/test_demo/ffmpeg_demo
/ ffmpeg_demo.c
Found pragma in SECTION
cim taskc 8, 33333, 1, hostflg, bufrdy
function name: ffmpeg_demo_cim_func_1

/root/Signalogic_2012v4/CIM/cimrt
##################################################
#Creating func_task files and editing _x86.c file#
##################################################
Read [ffmpeg_demo_cim_func_1.c]
 from parser logfile
Read [ffmpeg_demo_x86.c]
 from parser logfile
Currently processing file ffmpeg_demo_x86.c
Generating modified host file with name ffmpeg_demo_x86_rt.c
Region 1 has scalar variables :
  SOCKET hSocket;

About to add array variables in region 1 :
 8 8 8 8 8
###################################
#Starting Iterative build process#
###################################
#####################
#Generating Makefile#
#####################
#####################
#Generating .out file#
#####################
rm -rf ffmpeg_demo.out
rm -rf *.obj
rm -rf ./configPkg

Building file: ../cim.cfg
Invoking: XDCtools
/opt/ti/xdctools_3_25_05_94/xs --xdcpath="/opt/ti/bios_6_34_04_22/packages/;/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/packages/;/
opt/ti/ipc_1_24_00_16/packages;/opt/ti/pdk_C6678_1_0_0_17/packages;/opt/ti/ndk_2_22_00_06/packages;/opt/ti/framework_components_3_2
2_00_05/packages;/opt/ti/edma3_lld_02_11_03_02/packages;/opt/ti/xdais_7_20_00_07/packages" xdc.tools.configuro -o configPkg/ -t ti.
targets.elf.C66 -p SigC66xx -b /opt/ti/xdctools_3_25_05_94/packages/xdc/bld/config.bld -c /opt/ti/C6000CGT7.4.2/ ../cim.cfg
making package.mak (because of package.bld) ...
generating interfaces for package configPkg (because package/package.xdc.inc is older than package.xdc) ...
configuring cim.xe66 from package/cfg/cim_pe66.cfg ...
useModule Settings: 6600
Auto-registering resource manager for EDMA3CHAN
Inside EDMA3 RM getLibs
    will link with ti.sdo.edma3.rm:lib/c6678-evm/66/release/edma3_lld_rm.ae66
cle66 package/cfg/cim_pe66.c ...
Finished building: ../cim.cfg

Building object files.
Invoking: C6000 Compiler
/opt/ti/C6000CGT7.4.2/bin/cl6x -mv6600 -O3 -D_CIMF -D_BIOS_ -D_SIGC6XXX_ -D_SIGC66XX_ -D_SYSBIOS_ -DCHIP_6678 -DCHIP_C66XX -D_TI66X
```

Figure 4-37a, vid_streaming CIM build process output display

```
 -D_USE_CM_ -DC6600 -D_USE_RMAN_ -D_USE_IPC_ -D_USE_VID_ -D_USE_NDK_ -Dti_targets_elf_C66 -I/root/Signalogic_2012v4/TI_C6X/Voice/AM
R-WB/ -I/root/Signalogic_2012v4/TI_C6X/Voice/common/ -I/root/Signalogic_2012v4/mCPU_DSP/ -I/root/Signalogic_2012v4/mCPU_DSP/Video -
I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/qmss/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/cppi/ -I/root/Signalogic_2012v4/mCPU
_DSP/Voice/ -I/root/Signalogic_2012v4/mCPU_DSP/Host/ -I/root/Signalogic_2012v4/mCPU_DSP/CIM -I/root/Signalogic_2012v4/mCPU_DSP/C6xx
x/ -I/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/ -I/opt/ti/ndk_2_22_00_06/packages/ti/ndk/inc/ -I/opt/ti/xdctools_3_25_05_94/packa
ges/ -I/opt/ti/bios_6_34_04_22/packages/ -I/opt/ti/bios_6_34_04_22/packages/ti/bios/include/ -I/opt/ti/C6000CGT7.4.2/include/ -I/op
t/ti/C6000CGT7.4.2/lib/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ -I/opt/ti/xdais_7_20_00_07/packages -I/opt/ti/xdais_7_20_00_07/packa
ges/ti/xdais -I/opt/ti/xdais_7_20_00_07/packages/ti/xdais/dm -I/opt/ti/edma3_lld_02_11_03_02/packages -I/opt/ti/framework_component
s_3_22_00_05/packages -I/opt/ti/framework_components_3_22_00_05/packages/ti/sdo/fc/utils/api -I/opt/ti/ipc_1_24_00_16/packages -I/o
pt/ti/mcsdk_video_2_2_0_38/dsp/components --symdebug:dwarf --display_error_number --diag_warning=225 --mem_model:data=far --abi=eab
i --preproc_with_compile --cmd_file=./configPkg/compiler.opt  -@../buildRTAF.lst
[Analog_io.c]
[bsp.c]
[gpio.c]
[host.c]
[logic_uart.c]
[main.c]
[McBSP_Isr.c]
[profile.c]
[main_rtaf.c]
[rman_init.c]
[rtaf_hw_init.c]
[rtaf_periph.c]
[sd4_init.c]
[standalone.c]
[timer1_gpio.c]
[userproc.c]
[Bitrev.c]
[C6xxxLib.c]
[c_memory.c]
[Codec.c]
[event.c]
[Fft2.c]
[fs_conv.c]
[Isr0.c]
[Isr1.c]
[Isr2.c]
[Isr5.c]
[Isr6.c]
[Modul0.c]
[Modul1.c]
[Modul2.c]
[Modul3.c]
[Modul4.c]
[Modul5.c]
[Modul6.c]
[Modul7.c]
[Sintab.asm]
[t6xbeg.asm]
[t6xdef.asm]
[vectors.asm]
Finished building RTAF object files.

Building object files.
Invoking: C6000 Compiler
/opt/ti/C6000CGT7.4.2/bin/cl6x -mv6600 -O3 -D_CIMF -D_BIOS_ -D_SIGC6XXX_ -D_SIGC66XX_ -D_SYSBIOS_ -DCHIP_6678 -DCHIP_C66XX -D_TI66X
 -D_USE_CM_ -DC6600 -D_USE_RMAN_ -D_USE_IPC_ -D_USE_VID_ -D_USE_NDK_ -Dti_targets_elf_C66 -I/root/Signalogic_2012v4/TI_C6X/Voice/AM
R-WB/ -I/root/Signalogic_2012v4/TI_C6X/Voice/common/ -I/root/Signalogic_2012v4/mCPU_DSP/ -I/root/Signalogic_2012v4/mCPU_DSP/Video -
I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/qmss/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/cppi/ -I/root/Signalogic_2012v4/mCPU
_DSP/Voice/ -I/root/Signalogic_2012v4/mCPU_DSP/Host/ -I/root/Signalogic_2012v4/mCPU_DSP/CIM -I/root/Signalogic_2012v4/mCPU_DSP/C6xx
x/ -I/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/ -I/opt/ti/ndk_2_22_00_06/packages/ti/ndk/inc/ -I/opt/ti/xdctools_3_25_05_94/packa
ges/ -I/opt/ti/bios_6_34_04_22/packages/ -I/opt/ti/bios_6_34_04_22/packages/ti/bios/include/ -I/opt/ti/C6000CGT7.4.2/include/ -I/op
t/ti/C6000CGT7.4.2/lib/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ -I/opt/ti/xdais_7_20_00_07/packages -I/opt/ti/xdais_7_20_00_07/packa
ges/ti/xdais -I/opt/ti/xdais_7_20_00_07/packages/ti/xdais/dm -I/opt/ti/edma3_lld_02_11_03_02/packages -I/opt/ti/framework_component
s_3_22_00_05/packages -I/opt/ti/framework_components_3_22_00_05/packages/ti/sdo/fc/utils/api -I/opt/ti/ipc_1_24_00_16/packages -I/o
pt/ti/mcsdk_video_2_2_0_38/dsp/components --symdebug:dwarf --display_error_number --diag_warning=225 --mem_model:data=far --abi=eab
i --preproc_with_compile --cmd_file=./configPkg/compiler.opt  -@../buildH264.lst
[alg_create.c]
[alg_malloc.c]
[platform_osal.c]
[resourcemgr.c]
[h264hpvenc_ti_buffermanager.c]
[h264hpvenc ti config.c]
```

Figure 4-37b, vid_streaming CIM build process output display, cont.

```
[h264hpvenc_ti_mc_api.c]
[h264_encode.c]
[h264hpvenc_ti_testwrapper.c]
[h264hpvenc_ti_GateMPBarrier.c]
[h264hpvenc_ti_edmaConfig.c]
[h264hpvenc_ti_mcIpc.c]
[h264hpvenc_ti_shmemmanager.c]
[siuVigdkGmp.c]
Finished building H264 object files.

Building object files.
Invoking: C6000 Compiler
/opt/ti/C6000CGT7.4.2/bin/cl6x -mv6600 -O3 -D_CIMF -D_BIOS_ -D_SIGC6XXX_ -D_SIGC66XX_ -D_SYSBIOS_ -DCHIP_6678 -DCHIP_C66XX -D_TI66X
 -D_USE_CM_ -DC6600 -D_USE_RMAN_ -D_USE_IPC_ -D_USE_VID_ -D_USE_NDK_ -Dti_targets_elf_C66 -I/root/Signalogic_2012v4/TI_C6X/Voice/AM
R-WB/ -I/root/Signalogic_2012v4/TI_C6X/Voice/common/ -I/root/Signalogic_2012v4/mCPU_DSP/ -I/root/Signalogic_2012v4/mCPU_DSP/Video -
I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/qmss/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/cppi/ -I/root/Signalogic_2012v4/mCPU
_DSP/Voice/ -I/root/Signalogic_2012v4/mCPU_DSP/Host/ -I/root/Signalogic_2012v4/mCPU_DSP/CIM -I/root/Signalogic_2012v4/mCPU_DSP/C6xx
x/ -I/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/ -I/opt/ti/ndk_2_22_00_06/packages/ti/ndk/inc/ -I/opt/ti/xdctools_3_25_05_94/packa
ges/ -I/opt/ti/bios_6_34_04_22/packages/ -I/opt/ti/bios_6_34_04_22/packages/ti/bios/include/ -I/opt/ti/C6000CGT7.4.2/include/ -I/op
t/ti/C6000CGT7.4.2/lib/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ -I/opt/ti/xdais_7_20_00_07/packages -I/opt/ti/xdais_7_20_00_07/packa
ges/ti/xdais -I/opt/ti/xdais_7_20_00_07/packages/ti/xdais/dm -I/opt/ti/edma3_lld_02_11_03_02/packages -I/opt/ti/framework_component
s_3_22_00_05/packages -I/opt/ti/framework_components_3_22_00_05/packages/ti/sdo/fc/utils/api -I/opt/ti/ipc_1_24_00_16/packages -I/o
pt/ti/mcsdk_video_2_2_0_38/dsp/components --symdebug:dwarf --display_error_number --diag_warning=225 --mem_model:data=far --abi=eab
i --preproc_with_compile --cmd_file=./configPkg/compiler.opt  -@../cim_build.lst
Finished building cim object files.

Building object files.
Invoking: C6000 Compiler
/opt/ti/C6000CGT7.4.2/bin/cl6x -mv6600 -O3 -D_CIMF -D_BIOS_ -D_SIGC6XXX_ -D_SIGC66XX_ -D_SYSBIOS_ -DCHIP_6678 -DCHIP_C66XX -D_TI66X
 -D_USE_CM_ -DC6600 -D_USE_RMAN_ -D_USE_IPC_ -D_USE_VID_ -D_USE_NDK_ -Dti_targets_elf_C66 -I/root/Signalogic_2012v4/TI_C6X/Voice/AM
R-WB/ -I/root/Signalogic_2012v4/TI_C6X/Voice/common/ -I/root/Signalogic_2012v4/mCPU_DSP/ -I/root/Signalogic_2012v4/mCPU_DSP/Video -
I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/qmss/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/cppi/ -I/root/Signalogic_2012v4/mCPU
_DSP/Voice/ -I/root/Signalogic_2012v4/mCPU_DSP/Host/ -I/root/Signalogic_2012v4/mCPU_DSP/CIM -I/root/Signalogic_2012v4/mCPU_DSP/C6xx
x/ -I/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/ -I/opt/ti/ndk_2_22_00_06/packages/ti/ndk/inc/ -I/opt/ti/xdctools_3_25_05_94/packa
ges/ -I/opt/ti/bios_6_34_04_22/packages/ -I/opt/ti/bios_6_34_04_22/packages/ti/bios/include/ -I/opt/ti/C6000CGT7.4.2/include/ -I/op
t/ti/C6000CGT7.4.2/lib/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ -I/opt/ti/xdais_7_20_00_07/packages -I/opt/ti/xdais_7_20_00_07/packa
ges/ti/xdais -I/opt/ti/xdais_7_20_00_07/packages/ti/xdais/dm -I/opt/ti/edma3_lld_02_11_03_02/packages -I/opt/ti/framework_component
s_3_22_00_05/packages -I/opt/ti/framework_components_3_22_00_05/packages/ti/sdo/fc/utils/api -I/opt/ti/ipc_1_24_00_16/packages -I/o
pt/ti/mcsdk_video_2_2_0_38/dsp/components --symdebug:dwarf --display_error_number --diag_warning=225 --mem_model:data=far --abi=eab
i --preproc_with_compile --cmd_file=./configPkg/compiler.opt   ../ffmpeg_demo_func_task_1.c ../ffmpeg_demo_cim_func_1.c
[ffmpeg_demo_func_task_1.c]
[ffmpeg_demo_cim_func_1.c]
Finished building generated object files.

Building target: ffmpeg_demo.out
Invoking: Linker
/opt/ti/C6000CGT7.4.2/bin/cl6x -mv6600 -O3 -D_CIMF -D_BIOS_ -D_SIGC6XXX_ -D_SIGC66XX_ -D_SYSBIOS_ -DCHIP_6678 -DCHIP_C66XX -D_TI66X
 -D_USE_CM_ -DC6600 -D_USE_RMAN_ -D_USE_IPC_ -D_USE_VID_ -D_USE_NDK_ -Dti_targets_elf_C66 --display_error_number --diag_warning=225
 --mem_model:data=far --abi=eabi -z -mffmpeg_demo.map --warn_sections --display_error_number -i/opt/ti/C6000CGT7.4.2/lib/ -i/opt/ti
/C6000CGT7.4.2/include/ -i/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/ -i/opt/ti/pdk_C6678_1_0_0_17/packages/ -i/opt/ti/pdk_C6678_1
_0_0_17/packages/ti/transport/ndk/nimu/lib/  -i/root/Signalogic_2012v4/TI_C6X/Voice/AMR-WB/ -i/root/Signalogic_2012v4/mCPU_DSP/Vide
o -i/opt/ti/pdk_C6678_1_0_0_17/packages/ti/csl/lib -i/opt/ti/mcsdk_video_2_2_0_38/dsp/components/ti/mas/tsu/ce66 -i/opt/ti/mcsdk_vi
deo_2_2_0_38/dsp/components/ti/mas/util/ce66 --reread_libs --rom_model -offmpeg_demo.out *.obj ffmpeg_demo_func_task_1.obj ffmpeg_d
emo_cim_func_1.obj -l./configPkg/linker.cmd  -llibc.a -lh264hpvenc_ti.le66 -l/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/cppi/lib/ti
.drv.cppi.ae66 -l/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/qmss/lib/ti.drv.qmss.ae66 -l/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/
pa/lib/ti.drv.pa.ae66 -l/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/pcie/lib/ti.drv.pcie.ae66 -l/opt/ti/pdk_C6678_1_0_0_17/packages/
ti/platform/evmc6678l/platform_lib/lib/debug/ti.platform.evm6678l.ae66 -lti.csl.ae66 -l../t66linkSYSBIOS.cmd
<Linking>
Finished building target: ffmpeg_demo.out

/root/Signalogic_2012v4/CIM/cimrt
############################
#Generating HOST Executable#
############################
rm -rf *.o
rm -rf ffmpeg_demo
/usr/bin/gcc -I/root/Signalogic_2012v4/DirectCore/include  -O3 --std=c99 -D_CIM -D_SIGC66XX -D_LINUX_ -D_POSIX_C_SOURCE=200809L  -c
 ffmpeg_demo_cim_func_1.c -o ffmpeg_demo_cim_func_1.o
/usr/bin/gcc -I/root/Signalogic_2012v4/DirectCore/include  -O3 --std=c99 -D_CIM -D_SIGC66XX -D_LINUX_ -D_POSIX_C_SOURCE=200809L  -c
 ffmpeg_demo_x86_rt.c -o ffmpeg_demo_x86_rt.o
/usr/bin/gcc -I/root/Signalogic_2012v4/DirectCore/include  -O3 --std=c99 -D_CIM -D_SIGC66XX -D_LINUX_ -D_POSIX_C_SOURCE=200809L  ff
mpeg_demo_cim_func_1.o ffmpeg_demo_x86_rt.o  -lhwmgr -lhwlib -lpthread -lstdc++  -o ./ffmpeg_demo
cimrt$
```

Figure 4-37c, vid_straming CIM build process output display, cont.

After the CIM build process completes, a vid_streaming executable file will be created, and can be run as a typical Linux program:

```
  vid_streaming$ ./vid_streaming -m1 -f1250 -s2 -estream.out -cSIGC66XX-8
-i/install_path/Signalogic/video_files/airshow_352x288p_30fps_420fmt.yuv
-x352 -y288 -r30 -B1500000 -oencoded.h264
```

which should produce screen display output similar to Figure 4-38 below.

```
DSInitCores: qwCoreList = 0xff
SlotIndx = 0
Card_Type 0x8106
Talker not used
DSXInitCard SigC66x
ds66xInitCard:
InitCardT50 success
DSXInitCard Successful!
Loading input video .yuv data...
Clearing output video mem region...
Loading cores, core list = 0xff
DDR3 Time lapsed in microseconds : count 11878204  12
L2SRAM Time lapsed in microseconds : count 920  8
MSMCSRAM Time lapsed  in microseconds : count 740   11
DSXRunCard Successful!
Core 1 Finished
Core 2 Finished
Core 3 Finished
Core 4 Finished
Core 5 Finished
Core 6 Finished
Core 7 Finished
Core 0 Finished
Reading output video data, saving to file...
C66x testrun[0, 0x85f95c] = 9003
C66x cores error code[0x85ec28] = 0
numFramesEncoded[0x8597b4] = 40
vid_streaming$
```

Figure 4-38, vid_streaming demo output display

Note that on the vid_streaming folder, only the files vid_streaming.c and zero.dat are required. Except for the "encoded.h264" output file, which may be created when the demo executable code is run (as shown in Figure 4-38 above), all other files on the vid_streaming folder are auto-generated by the CIM build process.

4.7.3 Convolution demo

The CIM convolution demo is compatible with x86 OpenMP, c66x OpenMP, and the CIM® Hyperpiler™, which also uses OpenMP pragma syntax. The convolution demo is intended to (i) give a simple example of using CIM, (ii) show differences between the three platforms, and (iii) explain the theory behind the CIM Hyperpiler.

The CIM Hyperpiler generates source code streams for both x86 and c66x in order to build – from a single high level C/C++ program -- heterogeneous core run-time executables that coordinate and communicate with each other. Below is a source code excerpt from conv.c; the area marked in red shows use of #ifdefs to allow cross-platform compatibility.

```
int x[X_LEN + 2*H_LEN - 2] = { 0 };
int h[H_LEN] = { 0 };
int y[X_LEN + H_LEN - 1] = { 0 };

int h_len = H_LEN;
int x_len = X_LEN + 2*H_LEN - 2;
int conv_len = X_LEN + H_LEN - 1;

int sum;
int i,j;

int main (int argc, char *argv[]) {
/* use platform ifdef's to determine which OpenMP pragma keyword to use */

   #ifdef _CIM
   #pragma cim parallel for num_threads 5 private(i,j,sum) /* CIM combined x86 and c66x
   #else
   #pragma omp parallel for private(i,j,sum) /* x86 only or c66x only */
   #endif

   for (i=0; i<conv_len; i++) {  /* time domain convolution as 2-level for-loop */

      for (j=0; j<h_len; j++) sum += h[j] * x[i + j];

      y[i] = sum;
   }
/* Measure and report profiling results */
```

Figure 4-39, Convolution demo source code, with cross-platform compatibility #ifdefs highlighted

To run the CIM software convolution demo, first cd (change directory) to the conv folder:

```
cd /install_path/Signalogic/CIM/apps/test_demo/conv
```

Then enter:

```
make cim
```

Note that giving make with a cim target runs the following three (3) steps:

- CIM Hyperpiler process
- target CPU make
- standard Linux x86 make

Here is what the cim target looks like inside the Makefile:

```
./../../../cimrt/cimpp -px86 -ati66 -cSIGC66XX-32 -s
/root/Signalogic/CIM/apps/test_demo/conv/conv.c
```

Next, to run the convolution demo, enter:

```
./conv -m0x1f -f1400 -cSIGC66XX-8
```

 Copyright © Signalogic 2012-2015

The core list (`-m`), clock rate (`-f`), and card designator (`-c`) command line options are the same as described in DirectCore demo program sections above (e.g. boardTest, fftTest, streamTest, etc).

Here are some notes about running CIM generated executables:

1. Target CPU executables are automatically downloaded, initialized, and run. There is no ".out" file or other executable file entry in the command line (unlike DirectCore test programs that are run manually)

2. If you give a standard

   ```
   make
   ```

   command, then you are re-running the Linux make, not re-running the CIM process and regenerating x86 and target CPU source codes.

3. When giving the card designator in the command line, if you are running multiple process or sharing the card with other users, then it's necessary to pay attention to the number of cores requested. For example, the convolution demo command line above has the following card designator entry:

   ```
   -cSIGC66XX-32
   ```

   which requests 32 cores. If you enter:

   ```
   -cSIGC66XX-8
   ```

   then only 8 cores are requested

Figure 4-40 below is a screen grab showing expected output from the convolution demo.

 Copyright © Signalogic 2012-2015

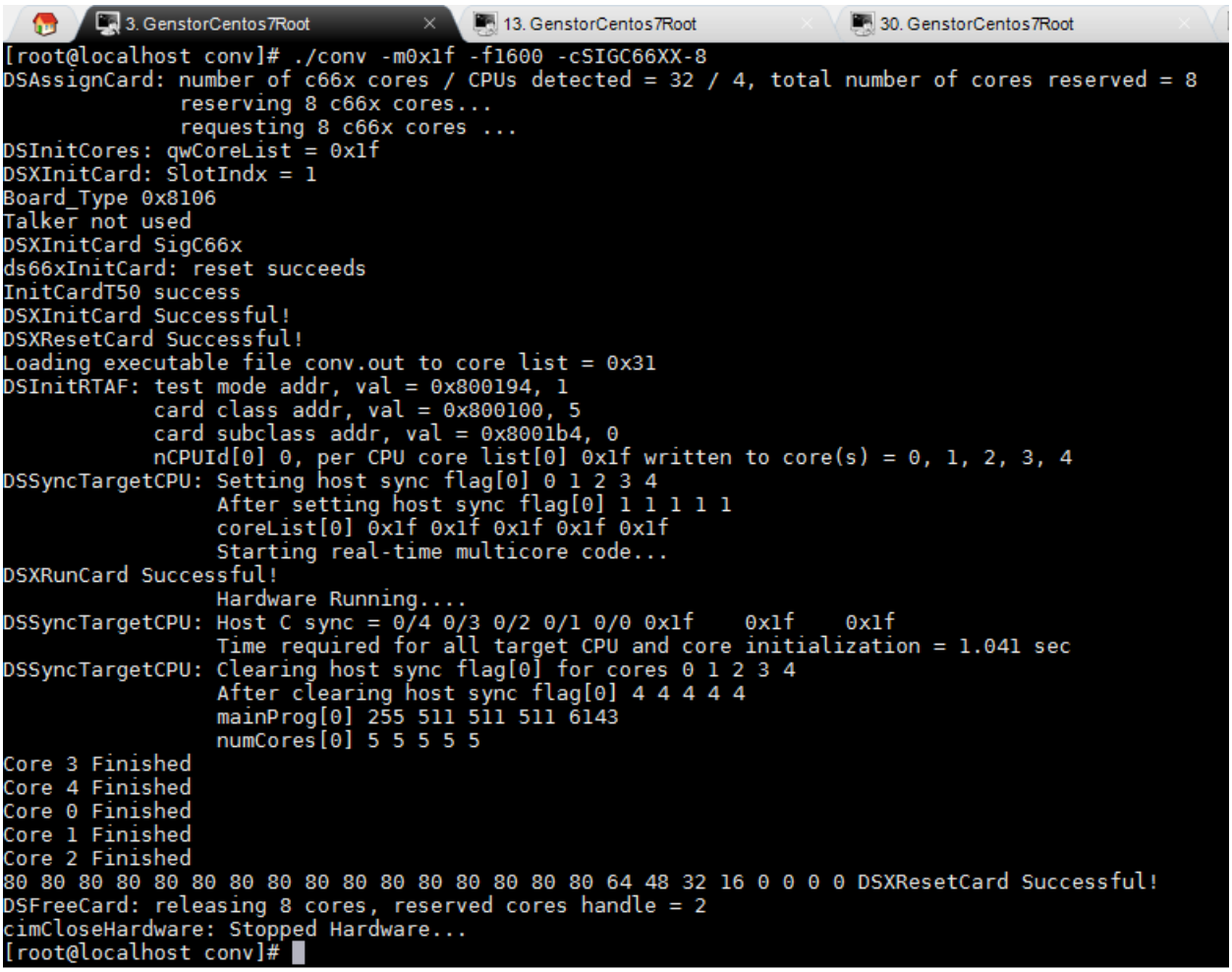

3. GenstorCentos7Root
13. GenstorCentos7Root
30. GenstorCentos7Root
[root@localhost conv]# ./conv -m0x1f -f1600 -cSIGC66XX-8
DSAssignCard: number of c66x cores / CPUs detected = 32 / 4, total number of cores reserved = 8
reserving 8 c66x cores...
requesting 8 c66x cores ...
DSInitCores: qwCoreList = 0x1f
DSXInitCard: SlotIndx = 1
Board_Type 0x8106
Talker not used
DSXInitCard SigC66x
ds66xInitCard: reset succeeds
InitCardT50 success
DSXInitCard Successful!
DSXResetCard Successful!
Loading executable file conv.out to core list = 0x31
DSInitRTAF: test mode addr, val = 0x800194, 1
card class addr, val = 0x800100, 5
card subclass addr, val = 0x8001b4, 0
nCPUId[0] 0, per CPU core list[0] 0x1f written to core(s) = 0, 1, 2, 3, 4
DSSyncTargetCPU: Setting host sync flag[0] 0 1 2 3 4
After setting host sync flag[0] 1 1 1 1 1
coreList[0] 0x1f 0x1f 0x1f 0x1f 0x1f
Starting real-time multicore code...
DSXRunCard Successful!
Hardware Running....
DSSyncTargetCPU: Host C sync = 0/4 0/3 0/2 0/1 0/0 0x1f 0x1f 0x1f
Time required for all target CPU and core initialization = 1.041 sec
DSSyncTargetCPU: Clearing host sync flag[0] for cores 0 1 2 3 4
After clearing host sync flag[0] 4 4 4 4 4
mainProg[0] 255 511 511 511 6143
numCores[0] 5 5 5 5 5
Core 3 Finished
Core 4 Finished
Core 0 Finished
Core 1 Finished
Core 2 Finished
80 80 80 80 80 80 80 80 80 80 80 80 80 80 80 64 48 32 16 0 0 0 0 DSXResetCard Successful!
DSFreeCard: releasing 8 cores, reserved cores handle = 2
cimCloseHardware: Stopped Hardware...
[root@localhost conv]#

Figure 4-40, Convolution demo output screen capture

 Copyright © Signalogic 2012-2015

5 pn4Test Host Program

Note: ***This section only applies to SigC641x DSP systems.***

The pn4Test host program controls and tests the McBSP1 and GPIO devices on the selected DSPs. During pn4Test operation, data is transmitted and received between selected DSPs, and real-time results are displayed in a Linux terminal window.

To build pn4Test, enter:

```
#cd /install_path/Signalogic/DirectCore/apps/SigC641x_C667x/pn4Test

#make
```

Running pn4Test with the command:

```
./pn4Test -h
```

will list the available command line options noting which options are mandatory for running pn4Test with a "+" sign. Supported command line options are listed below:

-d<0…2>	0	Detail display is disabled.
	1	Probe counters for each time the specified ISR (GPIO, TX, or RX) is entered are displayed on each display updated.
	2	FPGA revision numbers and memory symbol addresses are displayed once. Sync data for MCBSP and MCBSP_GPIO test modes is displayed once. A message indicating a signal was received with the DSP number that HINT was generated on is displayed each time the signal handler runs.
-f<N>	Sets the processor clock rate in MHz. (e.g. -f720 or -f1000)	
-m<bitmask>	Sets the processor select bit mask. Values can be entered in decimal or hex (e.g. –m17 is equivalent to –m0x11). Only one DSP from group A (DSPs 0-3) and only one DSP from group B (DSPs 4-7) should be selected for testing. If other settings are used, the first DSP from each group will be used, but the data display may not reflect the desired testing mode.	
-e<DSP COFF file name>	(e.g. -etmsc64xx_rtaf_ccs3.out)	

 Copyright © Signalogic 2012-2015

<table>
<tr><td>-c<designator></td><td colspan="3">Sets card designator (e.g. -cSIGC64XX)</td></tr>
<tr><td rowspan="5">-M<Test Mode></td><td>MCBSP</td><td colspan="2">Test only the serial mode sides of McBSP1 in serial mode</td></tr>
<tr><td>UART</td><td colspan="2">Test only the serial mode sides of McBSP1 in UART mode</td></tr>
<tr><td>GPIO</td><td colspan="2">Test only the GPIO sides of McBSP1</td></tr>
<tr><td>MCBSP_GPIO</td><td colspan="2">Test MCBSP and GPIO modes concurrently</td></tr>
<tr><td>UART_GPIO</td><td colspan="2">Test UART and GPIO modes concurrently</td></tr>
<tr><td rowspan="11">-r<Routing Config></td><td colspan="3">Routing configuration in FPGA logic. Sets the “PN4-McBSP Ctrl” register in FPGA #2. Takes a 32 bit value (e.g. –r0x300, the default value) with the following bit definitions:</td></tr>
<tr><td>31-11</td><td>Reserved</td><td></td></tr>
<tr><td rowspan="2">10</td><td rowspan="2">Group B Strobe Trigger Routing</td><td>0 = rev 2.4a</td></tr>
<tr><td>1 = rev 2.4b</td></tr>
<tr><td rowspan="2">9</td><td rowspan="2">PN4_FSX1_B direction</td><td>0 = output to PN4</td></tr>
<tr><td>1 = input from PN4</td></tr>
<tr><td rowspan="2">8</td><td rowspan="2">PN4_CLKX1_B direction</td><td>0 = output to PN4</td></tr>
<tr><td>1 = input from PN4</td></tr>
<tr><td rowspan="2">2</td><td rowspan="2">Group A Strobe Trigger Routing</td><td>0 = rev 2.4a</td></tr>
<tr><td>1 = rev 2.4b</td></tr>
<tr><td rowspan="2">1</td><td rowspan="2">PN4_FSX1_A direction</td><td>0 = output to PN4</td></tr>
<tr><td>1 = input from PN4</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">PN4_CLKX1_A direction</td><td>0 = output to PN4</td></tr>
<tr><td>1 = input from PN4</td></tr>
</table>

Note revision numbers and signal names (e.g. PN4_FSX1_B) are based on specifications shown in Figures 6-1 and 6-2, in section 5 below, “pn4Test Theory of Operation”.

 Copyright © Signalogic 2012-2015

After entering the mandatory command line options and running pn4Test, the entered options are echoed back to the terminal display. DSP, FPGA and on-chip memory tests are then run and DSPs are reset. Next, the SDRAM on each selected DSP is checked and results are displayed. Figure 5-1 below shows an example command line input followed by the first set of display and the SDRAM test results for DSP 0. This was run with the command:

```
# ./pn4Test -m0x11 -f1000 -etmsc64xx_rtaf_ccs3.out -cSIGC64XX -r0x300 -MGPIO -d2
```

root@localhost:~/Signalogic_2009v2/CIM_DSP/apps/SigC641x/pn4Test
localhost pn4Test # ./pn4Test -m0x11 -f1000 -otmsc64xx_rtaf_ccs3.out -b
SIGC64XX -r0x300 -MGPIO -d2
Starting PN4-McBSP1 test DSP program...
userIfs = {1, 17, 1000, 0, tmsc64xx_rtaf_ccs3.out, SIGC64XX, GPIO, 768,
0, 2, 0}
boardStr = SIGC64XX
Processor clock rate (Hz) = 1000000000
DSP COFF file = tmsc64xx_rtaf_ccs3.out
Test mode = GPIO
Routing config control register = 0x300
Initializing SigC64xx board...
DSInitProcessor: wProcessorList = 0x11
SlotIndx = 0
Board_Type 0x8806
Talker not used
DSXInitBoard SigC64x
ds64xInitBoard: Read REG_EXTx_SEL passed.
ds64xInitBoard: HPIA test passed
ds64xInitBoard: all DSP register and memory tests passed.
DSXInitBoard Successful!
Resetting all DSPs...
DSXResetBoard Successful!
SDRAM sanity check, DSP0
DSP_EMIFA_GCTL (0x1800000) = 0x1270c, configured as 0x1200c
DSP_EMIFA_CE0 (0x1800008) = 0xffffff30, configured as 0xffffff30
DSP_EMIFA_SDRAMCTL (0x1800018) = 0x52116000, configured as 0x53116000
DSP_EMIFA_SDRAMREF (0x180001c) = 0x580735, configured as 0x735
DSP_EMIFA_SDRAMEXT (0x1800020) = 0x54549, configured as 0x54549
number of SDRAM incorrect reads = 0
SDRAM location 0x81000000 = 0x7fff5567
number of SDRAM incorrect reads = 0
SDRAM location 0x81000004 = 0x7fff5568
number of SDRAM incorrect reads = 0
SDRAM location 0x81000008 = 0x7fff5569
number of SDRAM incorrect reads = 0
SDRAM location 0x8100000c = 0x7fff556a
number of internal SRAM incorrect reads = 0
int SRAM location 0x8000 = 0x7fff5567
number of internal SRAM incorrect reads = 0
int SRAM location 0x8004 = 0x7fff5568
number of internal SRAM incorrect reads = 0
int SRAM location 0x8008 = 0x7fff5569
number of internal SRAM incorrect reads = 0
int SRAM location 0x800c = 0x7fff556a

Figure 5-1, pn4Test initial display

 Copyright © Signalogic 2012-2015

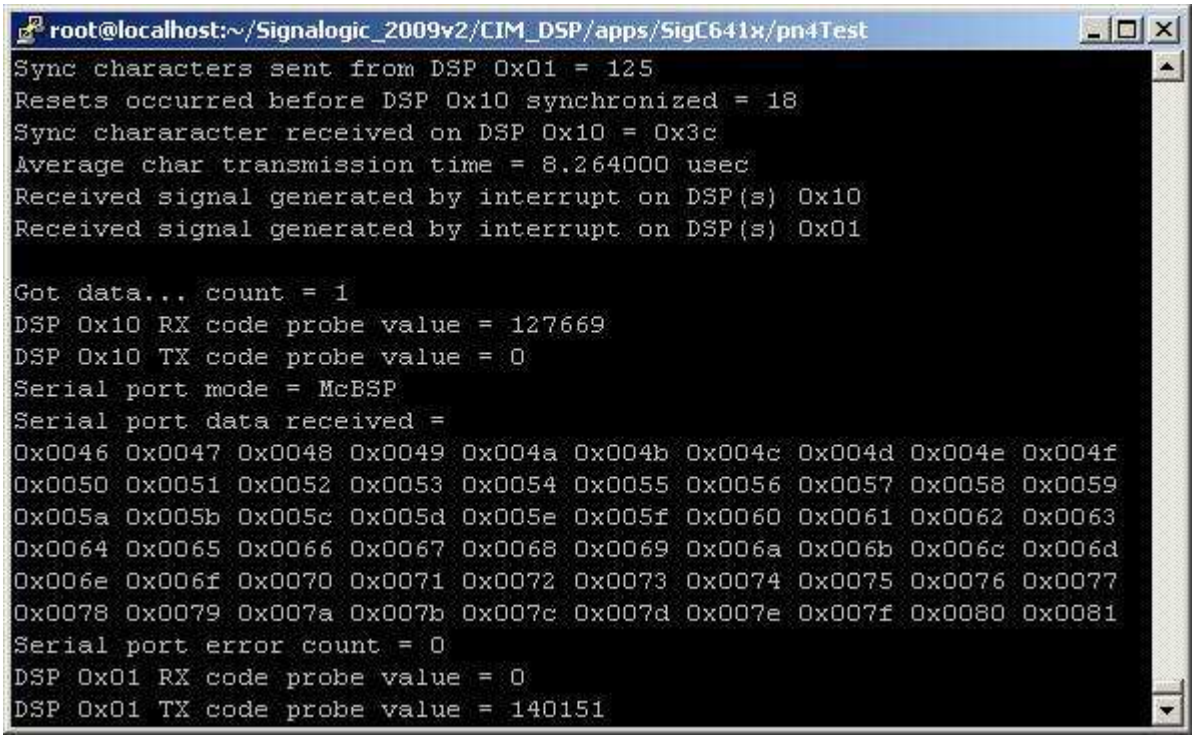
Once the SDRAM tests complete, if the details level is set at 2, FPGA logic versions are displayed followed by physical memory addresses (both on-chip and off-chip memory) of the DSP code variables that pn4Test will read from or write to for various settings and data display. If the test mode is configured as MCBSP or MCBSP_GPIO and the details level is set to 2, sync data for the receiving DSP (group B DSP) will be displayed. Lastly, repeating test data is displayed and continues to display until the program is stopped by pressing 'q' on the keyboard.

> ***Note:*** ***Using another method to terminate the program, such as Ctrl-C, may result in subsequent iterations of the program hanging or halting unexpectedly requiring the test to be run again and in some cases the driver to be reloaded.***

The serial port data counts up by 1 and wraps at 0xff for MCBSP test mode and wraps at 0xffff for UART test mode. This data is stored in a circular buffer with 64 elements. DSP code fills the buffer asynchronously from host reads so the current write position may be apparent by a jump in the numbers. When this occurs, the difference between these numbers should be 63. GPIO data consists of a counter for the number of strobe signals received and a display of the PWR_ENABLE_A signal (shown in Figure 5-1) data.

All data displayed also includes a corresponding error counter. Error counters are maintained by DSP code and indicate whether received data matches either the expected pattern for that signal or the specific data on that signal. The following figures show examples of the data display for different modes and different detail levels.

Figure 5-2 shows the MCBSP test mode sync data followed by the data display for MCBSP test mode. This is with details level set to 2.


root@localhost:~/Signalogic_2009v2/CIM_DSP/apps/SigC641x/pn4Test
Sync characters sent from DSP 0x01 = 125
Resets occurred before DSP 0x10 synchronized = 18
Sync chararacter received on DSP 0x10 = 0x3c
Average char transmission time = 8.264000 usec
Received signal generated by interrupt on DSP(s) 0x10
Received signal generated by interrupt on DSP(s) 0x01
Got data... count = 1
DSP 0x10 RX code probe value = 127669
DSP 0x10 TX code probe value = 0
Serial port mode = McBSP
Serial port data received =
0x0046 0x0047 0x0048 0x0049 0x004a 0x004b 0x004c 0x004d 0x004e 0x004f
0x0050 0x0051 0x0052 0x0053 0x0054 0x0055 0x0056 0x0057 0x0058 0x0059
0x005a 0x005b 0x005c 0x005d 0x005e 0x005f 0x0060 0x0061 0x0062 0x0063
0x0064 0x0065 0x0066 0x0067 0x0068 0x0069 0x006a 0x006b 0x006c 0x006d
0x006e 0x006f 0x0070 0x0071 0x0072 0x0073 0x0074 0x0075 0x0076 0x0077
0x0078 0x0079 0x007a 0x007b 0x007c 0x007d 0x007e 0x007f 0x0080 0x0081
Serial port error count = 0
DSP 0x01 RX code probe value = 0
DSP 0x01 TX code probe value = 140151

 Copyright © Signalogic 2012-2015

Figure 5-2, pn4Test MCBSP sync and data display

 Copyright © Signalogic 2012-2015

Figure 5-3 shows the data display for UART_GPIO test mode with the details level set to 2. Currently, there is a delay of 40 ms for UART mode before the first data word is sent. If DSP code is changed to remove this delay, errors may be seen on the data words during the first 40 ms of operation depending on when the first and subsequent data words are sent. Errors will not occur after the first 40 ms. This initialization delay compensates for differences in initialization code resulting in group A and B DSPs coming out of SYSBIOS initialization at different times.

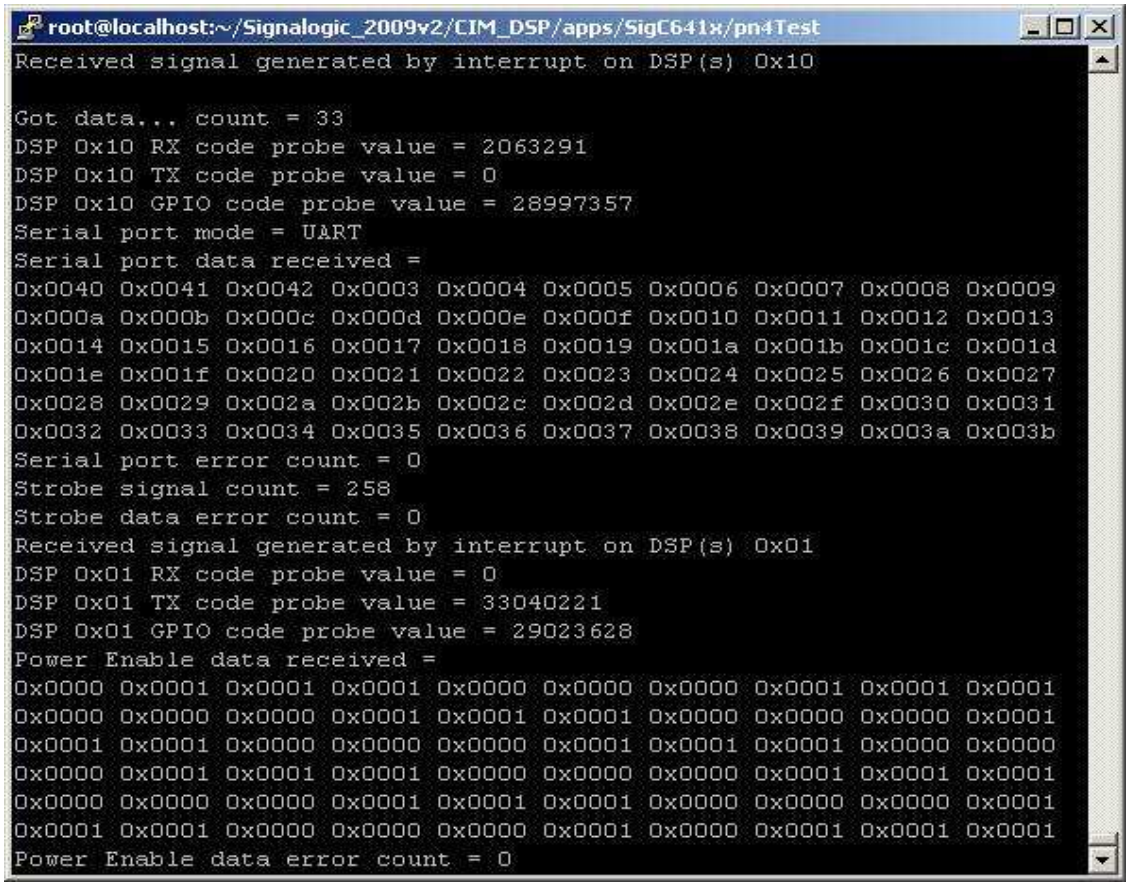


Figure 5-3, pn4Test –MUART_GPIO –d2 data display

Figure 5-4 shows the data display for UART test mode with the details disabled.

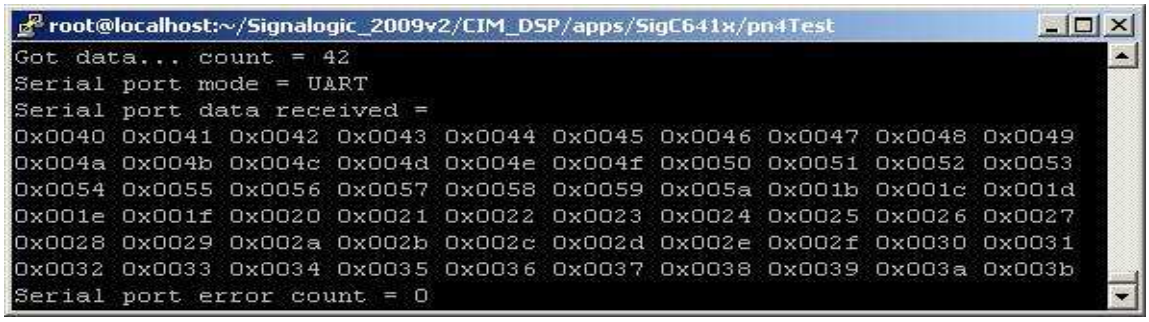


Figure 5-4, pn4Test –MUART –d0 data display

 Copyright © Signalogic 2012-2015

Figure 5-5 shows the data display for MCBSP_GPIO test mode with the details disabled.

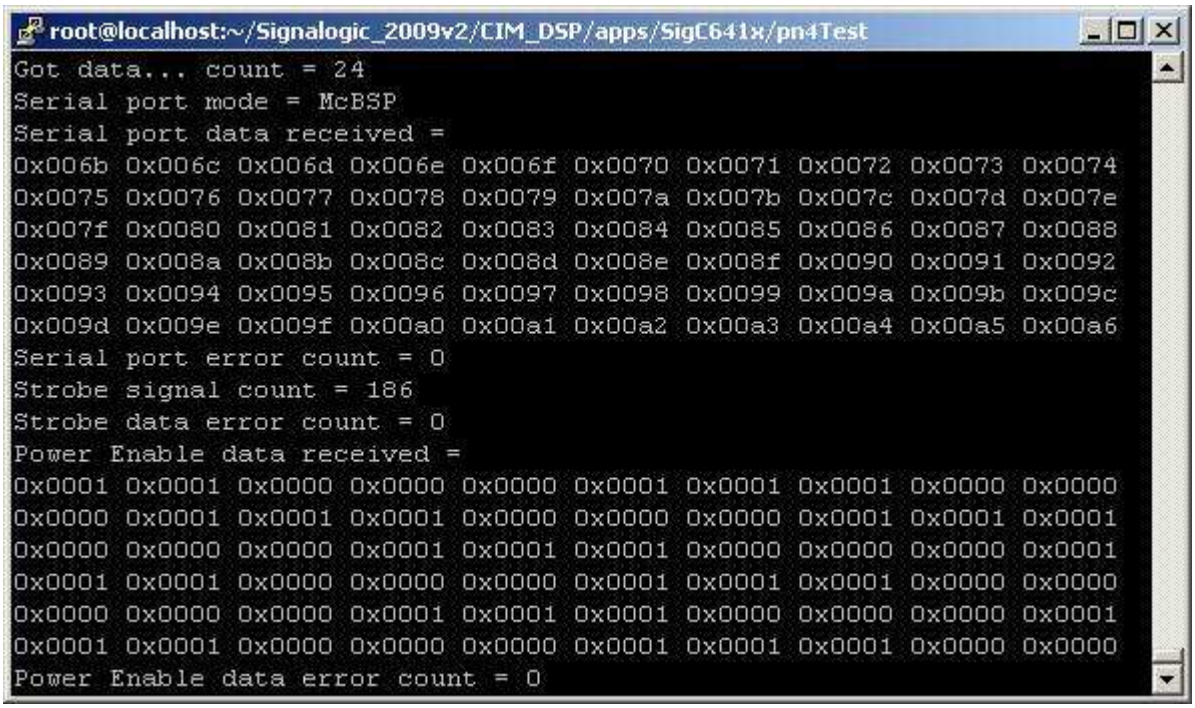


Figure 5-5, pn4Test –MMCBSP_GPIO –d0 data display

Figure 5-6 shows the data display for GPIO test mode with details level set to 1.

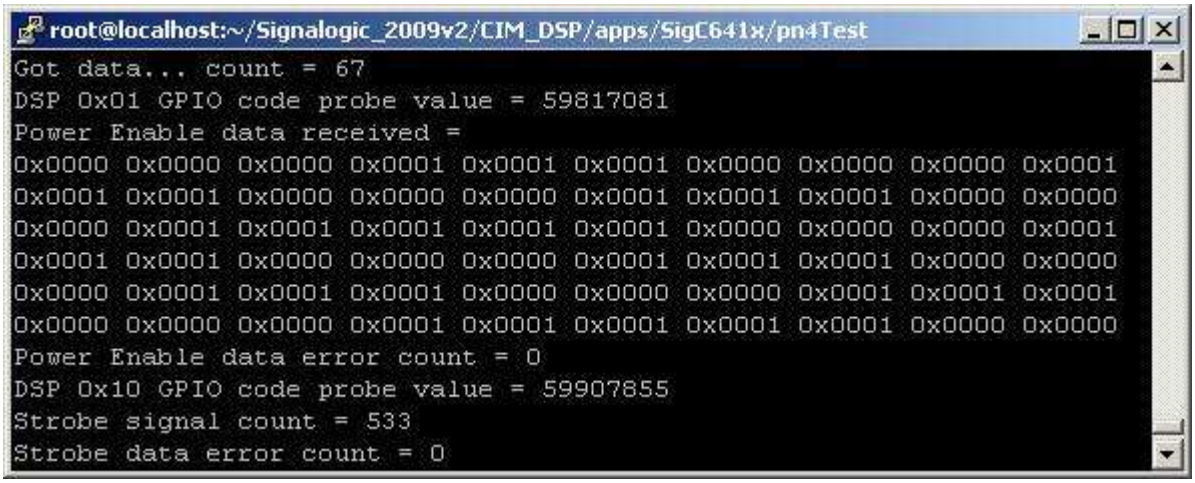


Figure 5-6, pn4Test –MGPIO –d1 data display

 Copyright © Signalogic 2012-2015

6 pn4Test Theory of Operation

Note: ***This section only applies to SigC641x systems.***

The current implementation of pn4Test runs according to the Rev 2.4a spec shown in Figure 6-1 on the next page. The Rev 2.4b spec can be seen on the subsequent page in Figure 6-2. Currently, Rev 2.4a is supported by host and DSP software and is the default mode of operation using command line option `-r0x300`. Rev 2.4b can be implemented with changes to the host and DSP software if the routing register in FPGA #2 is written appropriately with the command line option `-r` as outlined in section 3.

During the DSP initialization the test mode based on the entered command line parameter is set on the enabled DSPs. This determines which devices of the DSPs are initialized and at what rate. Table 5-1 below summarizes the rates used based on the test mode.

Test Mode	Group A DSP TX Rate	Group B DSP RX Rate	Serial Word Size	GPIO Sampling Rate
MCBSP	1 MHz	1 MHz	8-bit	N/A
UART	16 MHz	1 MHz	16-bit	N/A
GPIO	N/A	N/A	N/A	900 KHz
MCBSP_GPIO	1 MHz	1 MHz	8-bit	900 KHz
UART_GPIO	16 MHz	1 MHz	16-bit	900 KHz

Table 6-1, McBSP1 rates for each test mode

Note that in the specification diagrams in Figures 6-1 and 6-2 below, PN4 is a mating connector between SigC641x and PCIe adapter, and JP2 is the PCIe adapter external I/O connector (JP2 for Innovative Integration adapter, or P2 for Dynamic Engineering adapter).

 Copyright © Signalogic 2012-2015

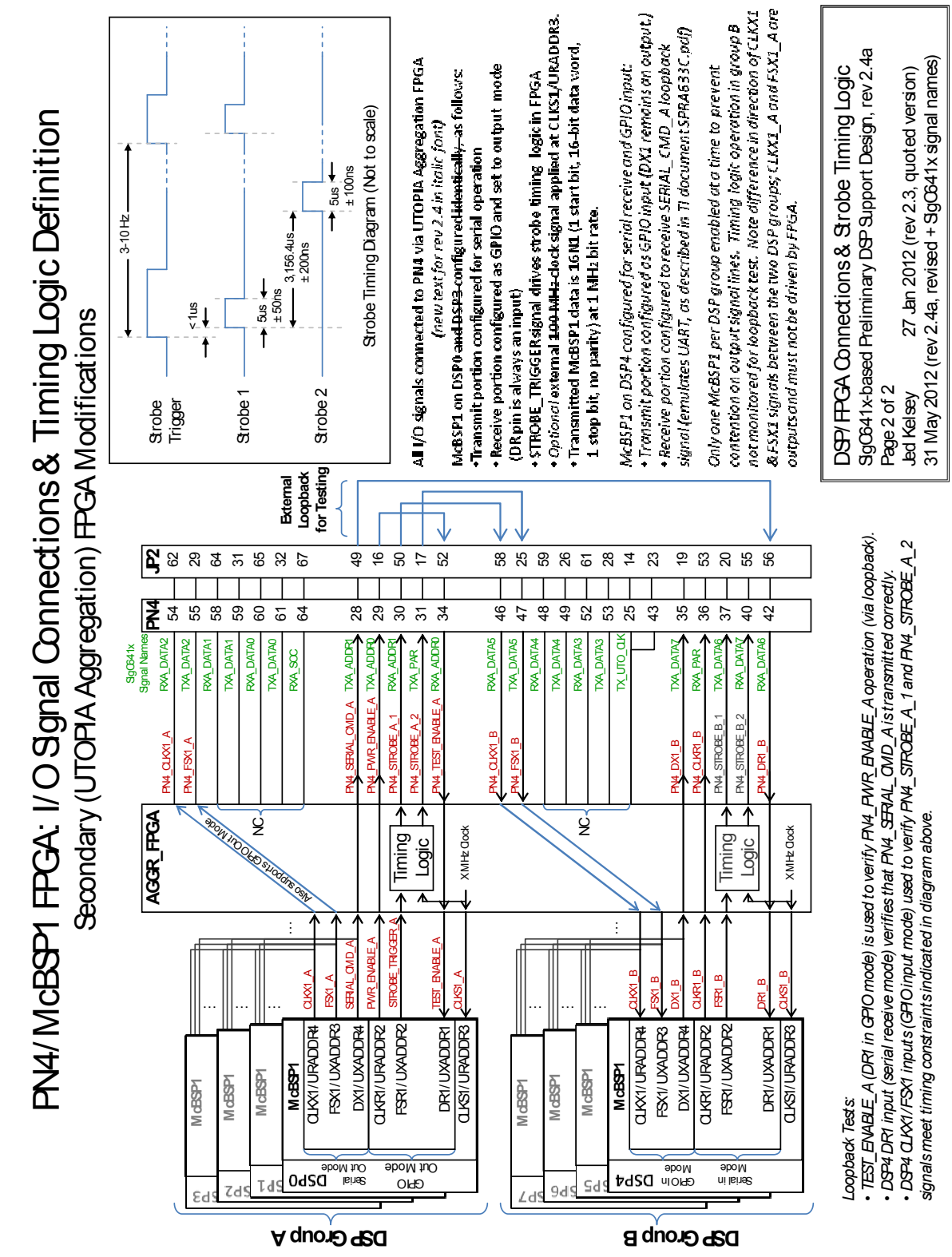

PN4/ McBSP1 FPGA: I/O Signal Connections & Timing Logic Definition
Secondary (UTOPIA Aggregation) FPGA Modifications
Strobe Trigger
Strobe 1
Strobe 2
3-10 Hz
<1us
5us
±50ns
3,156.4us
±200ns
5us
±100ns
Strobe Timing Diagram (Not to scale)
All I/O signals connected to PN4 via UTOPIA Aggregation FPGA
(new text for rev 2.4 in italic font)
McBSP1 on DSP0 and DSP3 configured identically, as follows:
• Transmit portion configured for serial operation
• Receive portion configured as GPIO and set to output mode (DR pin is always an input)
• STROBE_TRIGGER signal drives strobe timing logic in FPGA
• Optional external 100 MHz clock signal applied at CLKS1/URADDR3.
• Transmitted McBSP1 data is 16 bit (1 start bit, 16-bit data word, 1 stop bit, no parity) at 1 MHz bit rate.
McBSP1 on DSP4 configured for serial receive and GPIO input:
• Transmit portion configured as GPIO input (DX1 remains an output.)
• Receive portion configured to receive SERIAL_CMD_A loopback signal (emulates UART, as described in TI document SPRA633C.pdf)
Only one McBSP1 per DSP group enabled at a time to prevent contention on output signal lines. Timing logic operation in group B not monitored for loopback test. Note difference in direction of CLKX1 & FSX1 signals between the two DSP groups; CLKX1_A and FSX1_A are outputs and must not be driven by FPGA.
DSP/ FPGA Connections & Strobe Timing Logic
SgC641x-based Preliminary DSP Support Design, rev 2.4a
Page 2 of 2
Jed Kelsey 27 Jan 2012 (rev 2.3, quoted version)
31 May 2012 (rev 2.4a, revised + SgC641x signal names)
External Loopback for Testing
JP2
PN4
SgC641x Signal Names
AGGR_FPGA
Also supports GPIO Out Mode
NC
Timing Logic
XMHz Clock
DSP Group A
DSP Group B
McBSP1
DSP0 Serial Out Mode GPIO Out Mode
DSP4 Serial In Mode GPIO In Mode
Loopback Tests:
• TEST_ENABLE_A (DR1 in GPIO mode) is used to verify PN4_PWR_ENABLE_A operation (via loopback).
• DSP4 DR1 input (serial receive mode) verifies that PN4_SERIAL_CMD_A is transmitted correctly.
• DSP4 CLKX1/FSX1 inputs (GPIO input mode) used to verify PN4_STROBE_A_1 and PN4_STROBE_A_2 signals meet timing constraints indicated in diagram above.

Figure 6-1, Rev 2.4a Specifications

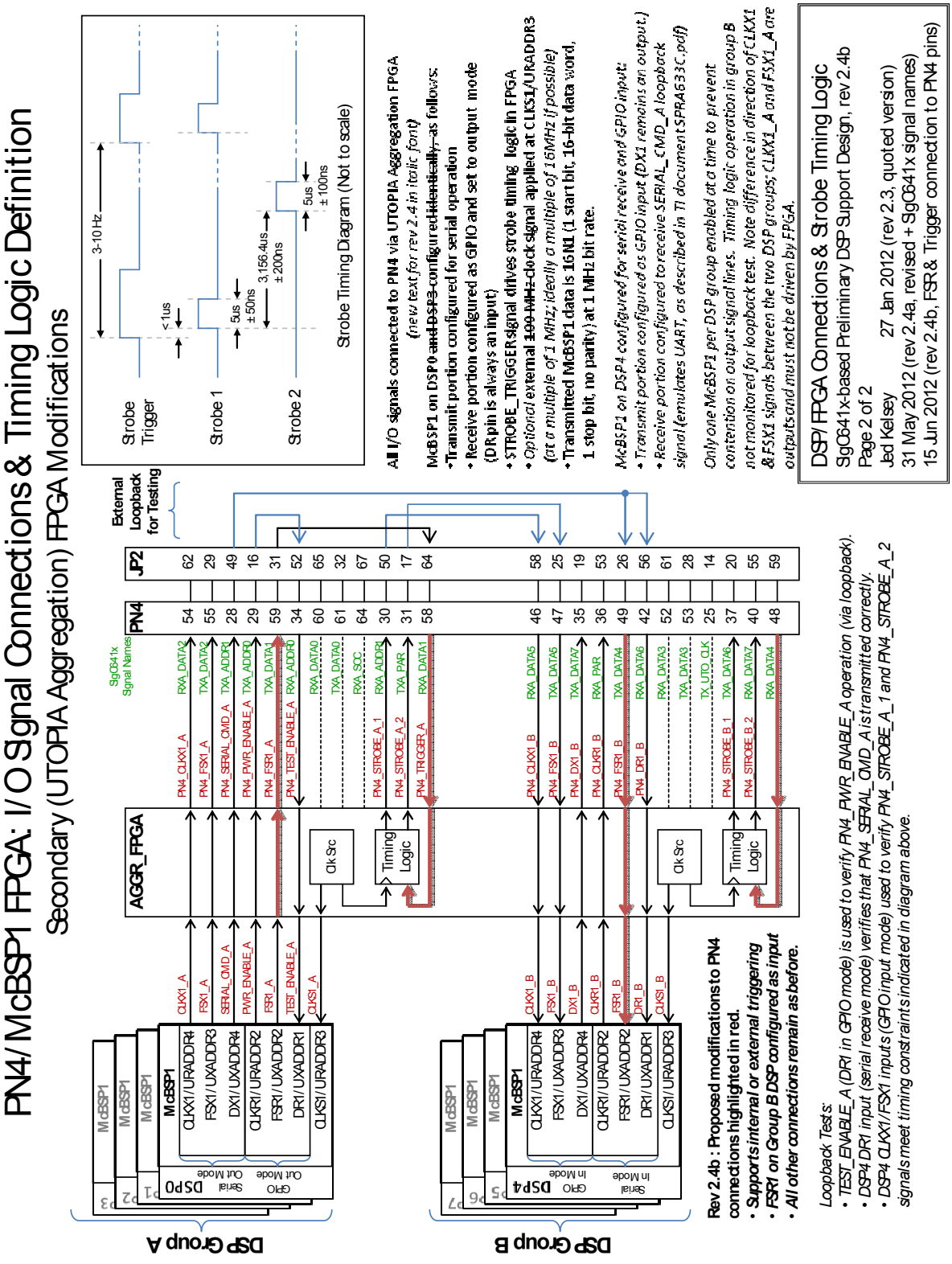

PN4/ McBSP1 FPGA: I/O Signal Connections & Timing Logic Definition
Secondary (UTOPIA Aggregation) FPGA Modifications
DSP/ FPGA Connections & Strobe Timing Logic
SgC641x-based Preliminary DSP Support Design, rev 2.4b
Page 2 of 2
Jed Kelsey 27 Jan 2012 (rev 2.3, quoted version)
31 May 2012 (rev 2.4a, revised + SgC641x signal names)
15 Jun 2012 (rev 2.4b, FSR & Trigger connection to PN4 pins)
Strobe Timing Diagram (Not to scale)
Strobe Trigger
Strobe 1
Strobe 2
3-10 Hz
<1us
5us
±50ns
3.156.4us
±200ns
5us
±100ns
All I/O signals connected to PN4 via UTOPIA Aggregation FPGA
(new text for rev 2.4 in italic font)
McBSP1 on DSP0 and DSP3 configured identically, as follows:
• Transmit portion configured for serial operation
• Receive portion configured as GPIO and set to output mode (DR pin is always an input)
• STROBE_TRIGGER signal drives strobe timing logic in FPGA
• Optional external 100 MHz clock signal applied at CLKS1/URADDR3 (at a multiple of 1 MHz; ideally a multiple of 16MHz if possible)
• Transmitted McBSP1 data is 16N1 (1 start bit, 16-bit data word, 1 stop bit, no parity) at 1 MHz bit rate.
McBSP1 on DSP4 configured for serial receive and GPIO input:
• Transmit portion configured as GPIO input (DX1 remains an output)
• Receive portion configured to receive SERIAL_CMD_A loopback signal (emulates UART, as described in TI document SPRA633C.pdf)
Only one McBSP1 per DSP group enabled at a time to prevent contention on output signal lines. Timing logic operation in group B not monitored for loopback test. Note difference in direction of CLKX1 & FSX1 signals between the two DSP groups; CLKX1_A and FSX1_A are outputs and must not be driven by FPGA.
External Loopback for Testing
JP2
PN4
AGGR_FPGA
SgC641x Signal Names
Clk Src
Timing Logic
DSP Group A
DSP Group B
McBSP1
GPIO Out Mode
Serial Out Mode
GPIO In Mode
Serial In Mode
DSP0
DSP4
Rev 2.4b : Proposed modifications to PN4 connections highlighted in red.
• Supports internal or external triggering
• FSR1 on Group B DSP configured as input
• All other connections remain as before.
Loopback Tests:
• TEST_ENABLE_A (DR1 in GPIO mode) is used to verify PN4_PWR_ENABLE_A operation (via loopback).
• DSP4 DR1 input (serial receive mode) verifies that PN4_SERIAL_CMD_A is transmitted correctly.
• DSP4 CLKX1/FSX1 inputs (GPIO input mode) used to verify PN4_STROBE_A_1 and PN4_STROBE_A_2 signals meet timing constraints indicated in diagram above.

Figure 6-2, Rev 2.4b Specifications

6.1 MCBSP Test Mode

In MCBSP test modes, the McBSPs are initialized to transmit and receive 8-bit words at a frequency of 1 MHz. The last step in initialization for the MCBSP test modes is syncing between group A and group B DSPs. Because of the fact that the McBSPs come out of reset at different times due to differences in initialization code and the sync signals are internally generated, group A and group B DSPs rarely send and receive at the same time initially. This results in the data received by the group B DSP being a shifted version of the data sent by the group A DSP. In order to resolve this problem, the group A DSP transmit needs to be synchronized with the group B DSP receive.

Sync takes place after McBSP1 is initialized on both DSPs. The group A DSP sends a sync character that cannot be duplicated through a shift of less than 8 bits (we use 0x3c as this character) for 1 ms. When the group B DSP receives a data word, it checks if the word matches the sync character. If the sync character was not received, McBSP1 is placed in reset for about an eighth of a bit period then re-enabled to check for another sync character. This continues until the expected sync character is received upon which the McBSP1 receive interrupt is enabled.

6.2 UART Test Mode

In UART modes, the McBSPs are initialized to transmit and receive 16-bit words with group A transmitting at 16 MHz and group B receiving at 1 MHz. Group A expands each bit sent to 16 bits in order for group B to receive each bit once due to the different frequencies. In this way, the word to be sent is oversampled by a factor of 16 before being sent so that group B receives the correct word that group A was intended to send.

6.3 GPIO Test Mode

In GPIO only mode, the McBSPs are initialized to transmit and receive at a frequency of 1 MHz. In all GPIO modes, GPIO sampling is done via a timer interrupt which occurs at a frequency of 900 KHz. This rate can be increased depending on the test mode used but ultimately has a limit based on peripheral register access speeds at which errors start to occur due to code delays caused by CPU stalls. 900 KHz is the highest rate at which all test modes reliably and consistently operate. An issue where SYSBIOS crashes arises when the SYSBIOS enables multiple interrupts concurrently causing multiple ISRs to be called at around the same time. As a workaround to this issue, initially the timer operates at a frequency of 16 Hz then reinitializes to the 900 KHz sampling rate after 1 sec of operation.

Due to the relatively low sampling rates of the strobe signals compared to their duration, verification of the strobe signal timing was measured with an oscilloscope and is shown in Figure 6-3 below.

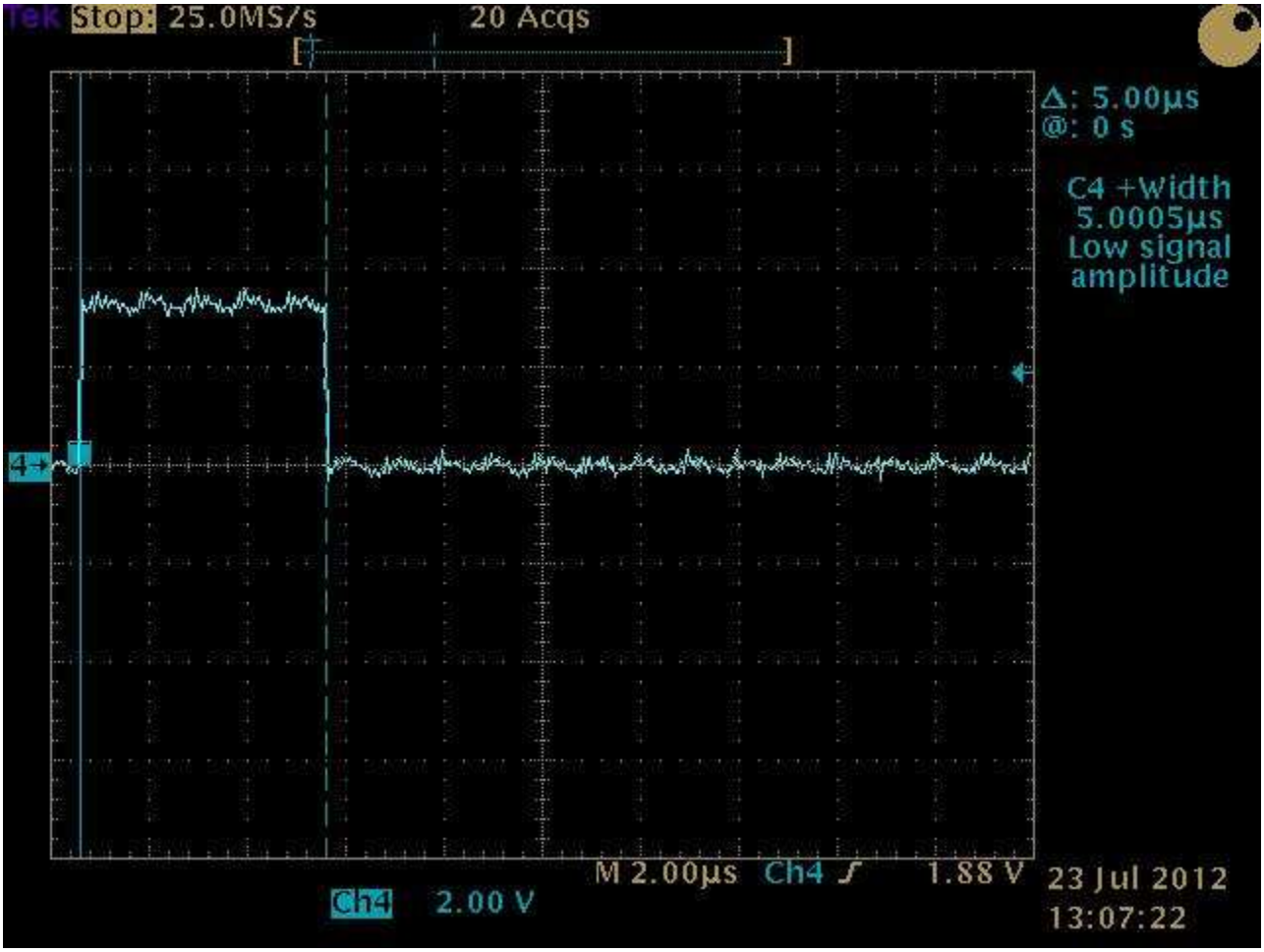


Figure 6-3, Oscilloscope capture of strobe signal

 Copyright © Signalogic 2012-2015

6.4 Host Operation

Once the board and enabled DSPs are reset and initialized, a callback function based on the uNotifyMode flag is registered. The uNotifyMode flag takes values 0-2 and can be modified in the pn4Test source code going into effect once the code is rebuilt. The possible uNotifyMode values with their descriptions are as follows:

0 – A timer is used to update the display and HINTs are ignored. The DSPs notify the host using memory locations and sync with the host using buffer numbers in memory. The DSPs do not generate HINTs in this mode.

1 – HINT notifications update the display. DSPs notify the host using HINT and sync with the host using buffer numbers in memory. DSP code waits for the host to update its buffer number. Both are operating on a dual-buffer (ring-buffer) basis. The DSPs generate HINTs in this mode.

2 – HINT notifications update the display. The DSPs notify the host using HINT and run free (no sync). DSPs continuously generate HINTs in this mode.

In all cases, a signal handler is created which calls the specified callback function. In the case of uNotifyMode 0, the signal handler runs on a received timer signal and calls the corresponding callback function which adds the DSP number to a queue (hereafter referred to as the DSP queue) if the host has been notified via memory locations by the DSP. In the cases of uNotifyMode 1 and 2, the signal handler runs whenever a HINT notification signal is received and calls the corresponding callback function which clears the HINT and adds the DSP number that the HINT arrived on to the DSP queue. DSP numbers in the DSP queue signify a pending DSP buffer data collection event. In the cases of uNotifymode 0 and 1, the corresponding callback function updates the host buffer number on the DSPs in order to sync with the DSP code.

After the callback function is registered, a keyboard thread and a display update thread are created. Both threads run until the keyboard thread sets an exit flag. The keyboard thread sleeps for 100ms, wakes up to checks for keyboard input and repeats. If a ‘q’ is read on the keyboard input, the thread sets the exit flag and terminates. The display update thread sleeps for 100ms and wakes up to check the exit flag and the DSP queue for pending DSP buffer data collection events. If a pending data collection event is found, data is read from the queued DSP number and displayed. The thread continues reading and displaying data until the queue is emptied upon which it sleeps for another 100 ms. The thread continues this process until the exit flag is set by the keyboard thread upon which it terminates.

Once the exit flag has been set and both threads terminate, hardware cleanup is performed which consists of unregistering the signal handler in the driver, clearing the board handle and setting a hardware ready flag to false. After hardware cleanup, pn4Test terminates.